Learning from Scenarios for Repairable Stochastic Scheduling

Kim van den Houten[1(✉)], David M. J. Tax[1], Esteban Freydell[2], and Mathijs de Weerdt[1]

[1] Delft University of Technology, Delft, The Netherlands
k.c.vandenhouten@tudelft.nl
[2] DSM-Firmenich, Delft, Netherlands

Abstract. When optimizing problems with uncertain parameter values in a linear objective, decision-focused learning enables end-to-end learning of these values. We are interested in a stochastic scheduling problem, in which processing times are uncertain, which brings uncertain values in the constraints, and thus repair of an initial schedule may be needed. Historical realizations of the stochastic processing times are available. We show how existing decision-focused learning techniques based on stochastic smoothing can be adapted to this scheduling problem. We include an extensive experimental evaluation to investigate in which situations decision-focused learning outperforms the state of the art, i.e., scenario-based stochastic optimization.

Keywords: Stochastic Scheduling · Repair · Decision-focused learning

1 Introduction

Decision-making can be challenging due to the stochastic nature of real-world processes. This complexity is evident in various contexts, such as manufacturing, where uncertain processing times make it challenging to meet strict customer deadlines. Formulating Constrained Optimization (CO) models for these problems is common, but unknown parameter values during decision-making add challenges, because wrong estimates of the parameters can lead to infeasibilities.

In practice, such infeasibilities are repaired when reality unfolds. For instance, in a manufacturing system, tasks may be postponed due to delays in earlier stages to maintain the factory's flow. Various repair policies and schedule definitions are used across different contexts.

Historical data, represented as scenarios of unknown parameters like task duration, are often available. Simple averaging of these scenarios is a common yet naive approach that ignores uncertainty. Stochastic programming [16] and robust optimization [1] offer alternatives, each with its challenges, such as scalability and too conservative solutions. Moreover, modeling realistic repair possibilities exactly is not always possible in such two-stage optimization approaches.

B. Dilkina (Ed.): CPAIOR 2024, LNCS 14743, pp. 234–242, 2024.
https://doi.org/10.1007/978-3-031-60599-4_15

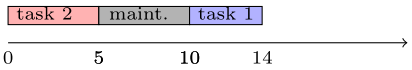


Fig. 1. Deterministic opt. schedule

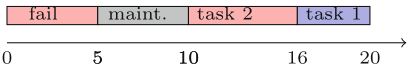


Fig. 2. Repair action when $y_2 = 6$

Decision-focused learning (DFL), extensively reviewed by [13], introduces a novel paradigm for stochastic optimization. This approach embeds an optimization model, like Constraint Programming (CP), in a training procedure to minimize a regret loss [6]. Challenges arise in backpropagation through combinatorial optimization problems, where solutions may change discontinuously. Recent research, including the score-function approach by [17], shows promising directions for handling uncertainty in constraints. Both exploring DFL with uncertainty in constraints, and analyzing the applicability of the score-function method are highlighted as valuable directions for further research [13].

In this research, we explore various scenario-based approaches for stochastic scheduling. The contribution is threefold: 1) we apply DFL for the first time to a repairable stochastic scheduling problem with stochastic processing times, 2) we demonstrate how an existing DFL technique that uses stochastic smoothing can be used to serve a stochastic scheduling problem where historical realizations of processing times are used, and 3) we include an extensive experimental evaluation in which we assess differences in performance between deterministic, stochastic programming, and a DFL approach.

2 Scheduling with Repair

We illustrate the effect of uncertainty in constraints with an example of scheduling two tasks on a single machine, where the average task lengths are $\bar{y}_1 = 4$, and $\bar{y}_2 = 5$. The machine is not available from $t = 5$ to $t = 10$. The task is to minimize makespan. Using the mean values, the optimal decision is to schedule first task 2, and then task 1, which gives us a makespan of 14 (see Figure 1), while scheduling first task 1, and then task 2 results in a makespan of 15.

Now suppose that task 1 is deterministic, and task 2 is stochastic, following the discrete uniform distribution $y_2 \sim U(\{3, 4, 5, 6, 7\})$. It still holds that the expected task lengths are $\bar{y}_1 = 4$, and $\bar{y}_2 = 5$. When we have $y_2 = 6$ and we schedule task 2 first, the effect of the repair strategy can be seen in Figure 2 and leads to a makespan of 20. Considering this repair, we can compute the expected values of the two alternative decisions and find that $\mathbb{E}$[first task 1, then task 2] $= 15$ and $\mathbb{E}$[first task 2, then task 1] $= 16.6$. So, considering the underlying distributions, it is better to first schedule task 1, instead of task 2. We observe that just using the expected values to come to a decision is not always a good idea when processing times are uncertain.

3 Decision-Focused Learning

Problem Setting. The goal is to optimize an optimization (e.g. scheduling) problem $z^*(y) = \arg\min_z f(z, y)$ *s.t.* $z \in C(y, z)$, where $f(z, y)$ is the objective

function given parameters y and decision z and the constraint set $C(y, z)$. However, the parameters (e.g. processing times) y are unknown at the time of solving. We are given a data set $\mathcal{D} = \{y_i\}_{i=1}^{n}$ with historical data on y. A common approach is to use the sample averages of $\bar{y}$ to solve the deterministic model and obtain $z^*(\bar{y})$ (which possibly requires reparations when the true values become known). Alternatively, we could take inspiration from the literature on DFL.

Decision-Focused Learning. The idea is to predict the unknowns $\widehat{y} = h_\theta(\mathcal{D})$ based on the data such that the task loss is minimized. Since the unknown parameters occur in the constraints, predicted decisions must sometimes be corrected using a repair function (such as illustrated in Sect. 2). A common task loss for problems with unknown parameters in the constraints is the so-called post-hoc regret $PRegret$ loss [9], defined as:

$$PRegret(\widehat{y}, y) = f(z_{corr}(\widehat{y}, y), y) - f(z^*(y), y) + pen(z^*(\widehat{y}), z_{corr}(\widehat{y}, y)), \quad (1)$$

where y are the true coefficient values, $\widehat{y}$ are the predicted values, $z^*(\widehat{y})$ is the decision based on predicted values, and $z^*(y)$ is the optimal decision with perfect information such as defined by [3]. Then, we have $f(z^*(\widehat{y}), y)$, which are the costs for predicted decisions, and $f(z^*(y), y)$, which are true optimal costs. Due to uncertain parameters in the constraints, predicted decisions must sometimes be corrected using a repair function such that $z^*(\widehat{y}) \rightarrow z_{corr}(\widehat{y}, y)$. How this reparation is penalized is reflected in $pen(z^*(\widehat{y}), z_{corr}(\widehat{y}, y))$.

Zero-Gradient Problem. DFL procedures minimize the post-hoc regret loss by gradient-based optimization with respect to θ to optimize the prediction $\widehat{y} = h_\theta(\mathcal{D})$. However, this loss gives a zero-gradient problem because a combinatorial optimization solver is embedded in the loss computation [6], which is the $\frac{\delta z_{corr}(\widehat{y}, y)}{\delta \widehat{y}}$ term in (2).

$$\frac{\delta PRegret(\widehat{y}, y)}{\delta \theta} = \frac{\delta PRegret(z_{corr}(\widehat{y}, y), y)}{\delta z_{corr}(\widehat{y}, y)} \frac{\delta z_{corr}(\widehat{y}, y)}{\delta \widehat{y}} \frac{\delta \widehat{y}}{\delta \theta} \quad (2)$$

Stochastic Smoothing. A novel approach by Silvestri et al. [17] shows that this zero-gradient problem can be solved with a stochastic smoothing trick. The crux is to use a stochastic estimator $\widehat{y} \sim p_\theta(y)$ (where $p_\theta(y)$ is a parameterized distribution) instead of the point estimator $\widehat{y} = h_\theta(\mathcal{D})$. Using a stochastic estimator makes the loss function an expectation, for which the gradient can be approximated with the score-function gradient estimator (also known as likelihood ratio gradient estimator [7]) that uses:

$$\nabla_\theta \mathbb{E}_{\widehat{y} \sim p_\theta(y)}[PRegret(\widehat{y}, y)] = \mathbb{E}_{\widehat{y} \sim p_\theta(y)}[PRegret(\widehat{y}, y) \nabla_\theta \log(p_\theta(\widehat{y}))] \quad (3)$$

for which the derivation can be found in [17]. The most important assumption on $p_\theta(y)$ is that the probability density function must be differentiable with respect to θ. The right-hand side can be approximated with a Monte-Carlo method [15]. This score-function gradient estimation approach is also the foundation of the REINFORCE algorithm [19], and various other reinforcement learning algorithms [18]. How we exactly apply these techniques to our stochastic scheduling problem is explained in the next section.

4 From Scenarios to Schedules

Algorithm: We adapt DFL to align with our scheduling problem in Algorithm 1. The data $\mathcal{D} = \{y_i\}_{i=1}^{n}$ comprises historical examples of processing times y. We aim to learn which predictor $\widehat{y}$ minimizes the post-hoc regret. For gradient computation, we use a stochastic estimator parameterized by θ, for which a common choice is the Normal distribution [18]. During training, we sample $\widehat{y} \sim \mathcal{N}(\mu = \theta_\mu \cdot \bar{y}, \sigma = \theta_\sigma \cdot \bar{\sigma})$, where both μ and σ are trainable, and initially set to the sample average $\bar{y}$ and sample standard deviation $\bar{\sigma}$. In each training step, we sample a point y_i and a prediction $\widehat{y}$, compute schedule $z^*(\widehat{y})$, and update θ using the score-function gradient estimator that is provided in equation (3). After training, the stochastic estimator is treated as a point estimator by using $\widehat{y} = \mu$. Note that the distribution is only needed during training for gradient computation on the regret loss.

Example: We explain the zero-gradient problem and smoothing technique with our example from Sect. 2. Suppose $(y_1, y_2) = (4, 6)$, but y_2 is unknown. Figure 3 shows how $\widehat{y}_2$ affects the regret, which is the line with a discontinuity at $\widehat{y}_2 = 5$, indicating a jump in scheduling priority. The blue curves show different stochastic estimators, and the small circles the expected regret values when we sample $\widehat{y}_2$ from each distribution. The line through the circles represents the smoothened expected regret. We assess the applicability of Algorithm 1 using this example. The training data has an underlying distribution with $y_1 = 4$ and $y_2 \sim U(3, 4, 5, 6, 7)$. We expect the algorithm to find scaling $\theta_2 > 1$ to prioritize scheduling task 1. A small experiment confirms that regret drops when μ_1 is above five as anticipated, see Fig. 4.

Algorithm 1: DFL

Require: $\mathcal{D}_{train} = \{y_i\}_{i=1}^{n_{train}}$, $\mathcal{D}_{test} = \{y_i\}_{i=1}^{n_{test}}$
Initialize $\widehat{y} \sim p_\theta(\widehat{y})$ such that
$\widehat{y} \sim \mathcal{N}(\mu = \theta_\mu \cdot \bar{y}, \sigma = \theta_\sigma \cdot \bar{\sigma})$
for each epoch **do**
  for each batch in $\mathcal{D}_{train}$ **do**
    for each instance $(y_i, z^*(y_i))$ in batch **do**
      Sample $\widehat{y}$ from $p_\theta(\widehat{y})$
      Pass $\widehat{y}$ to solver to get schedule
      Compute post-hoc regret$(\widehat{y}, y_i)$
    end for
    Update θ with score-function:
    $\theta = \theta - \text{lr} \cdot \nabla_\theta PRegret(\widehat{y}, y_i) \nabla_\theta \log(p_\theta(\widehat{y}))$
  end for
end for
Pass $\widehat{y} = \mu$ to solver to get schedule
Evaluate post-hoc regret on $\mathcal{D}_{test}$

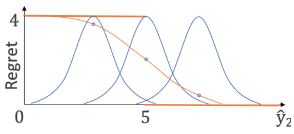


Fig. 3. Smoothing

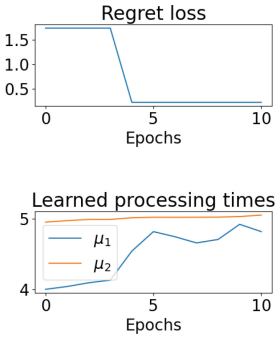


Fig. 4. Training curves

5 Experimental Evaluation

To understand the potential of DFL, we compare performance to deterministic and stochastic programming formulations (Sect. 5.2). We hypothesize that both stochastic programming and DFL outperform the more naive deterministic approach. Furthermore, we explore when DFL or stochastic programming performs better; we expect that DFL has better scalability to larger instances.

5.1 Problem Instances and Evaluation

We study a variant of the **Stochastic Resource Constraint Project Scheduling Problem** (RCPSP). We use two subsets of the PSPLib instances (being j301_1 to j303_10, and j901_1 -to j903_10) [11]. Furthermore, we use two sets of industry-inspired problem instances (small and large instances[1]), related to the factory of our industrial partner DSM-Firmenich. The data describing these instances is provided in our repository [8]. Originally deterministic, these instances are transformed into stochastic versions by sampling task durations from Normal distributions with mean d_j and standard deviation $\sqrt{d_j}$, where d_j is the deterministic processing time of task j from the original instance. We define a scenario as a processing time vector realization for one problem instance. For each instance, three datasets are created: one with 100 training scenarios, another with 50 for validation and tuning, and a final set of 50 for evaluation.

The **evaluation** approach evaluates first-stage start times z_j decisions based on the schedule makespan. Tasks unable to start due to resource constraints or precedence relations undergo a repair policy with (multiple) one-unit time postponements resulting in corrected start times z_j^{corr}. A penalty function measures the sum of start time deviations for all tasks j:

$$\text{pen}(z^*(\widehat{y}), z^*_{corr}(\widehat{y}, y)) = \rho \cdot \sum_{j \in J} z_j^{corr}(\widehat{y}, y) - z_j(\widehat{y}). \tag{4}$$

Here, ρ is the penalty coefficient, which we vary across experiments. Evaluation is conducted using the SimPy discrete-event simulation Python package [14].

5.2 Baseline Methods

We study problem cases where historical realizations of stochastic processing times are available (without feature data). This section describes two other scenario-based methods that are included in the experiments.

Deterministic Approach. This is a simple baseline, where we compute scenario averages of the unknown optimization coefficients. The deterministic constraint programming (CP) model that uses these averages uses the following nomenclature: J: set of all tasks, R: set of all resources, S_j: set of successors

[1] 25 instances with 40 to 480 tasks, 13 resource groups with different capacities.

of task j, j: subscript for tasks, r: subscript for resources, *parameters:* y_j: processing time of task j, $r_{r,j}$: resource requirement for task j, b_r: max capacity of resource r, $\text{minLag}_{j,i}$: min. difference between start times of tasks j and i, if i is a successor of j and *decision variables:* x_j: interval length for task j. The CP model is:

$$\text{Minimize } Makespan \qquad \text{s.t.} \tag{5a}$$

$$\text{Max}(\text{end\_of}(\text{x}_\text{j})) \leq Makespan \qquad j \in J \tag{5b}$$

$$\begin{aligned} \text{startOf}(\text{x}_\text{i}) \geq \text{endOf}(\text{x}_\text{j}); \qquad & \forall j \in J\ \forall i \in S_j \quad or \\ \text{startOf}(\text{x}_\text{i}) \geq \text{minLag}_{\text{j,i}} + \text{startOf}(\text{x}_\text{j}); \qquad & \forall j \in J\ \forall i \in S_j \end{aligned} \tag{5c}$$

$$\sum_{j \in J} \text{Pulse}(\text{x}_\text{j}, \text{r}_{\text{r,j}}) \leq \text{b}_\text{r} \qquad \forall r \in R \tag{5d}$$

$$x_j : \text{IntervalVar}(\text{J}, \text{y}_\text{j}) \qquad \forall j \in J \tag{5e}$$

In this model, (5b) defines the makespan which should be larger than the finish time of all tasks, and (5c) enforces precedence constraints between two tasks, where the $\text{minLag}_{\text{a,b}}$ is the minimal time difference needed between task a and b which is used for the industry instances. The CP pulse constraint (5d) models shared resource usage [4].

Stochastic Programming. The second baseline comprises a scenario-based stochastic programming formulation (again CP). The repair action is added to the stochastic model which comprises the possibility to postpone activities, together with a penalty term for the deviations from the earliest-start-time decision that is included in the objective. Note that we use the same nomenclature as for the deterministic model, but we introduce the notion of scenarios $\omega \in \Omega$, and the first-stage earliest-start-time decision variable $z_j\ \forall j \in J$.

$$\text{Min} \frac{1}{|\Omega|} \sum_{\omega \in \Omega} Makespan(\omega) + \rho \cdot \sum_{\omega \in \Omega} \sum_{j} \text{startOf}(\text{x}_\text{j}(\omega)) - z_j \quad \text{s.t} \tag{6a}$$

$$\text{Max}_\text{j}(\text{end\_of}(\text{x}_\text{j}(\omega))) \leq Makespan(\omega) \qquad \forall \omega \in \Omega \tag{6b}$$

$$\begin{aligned} \text{startOf}(\text{x}_\text{i}(\omega)) \geq \text{endOf}(\text{x}_\text{j}(\omega)); \qquad & \forall j \in J\ \forall i \in S_j \quad or \\ \text{startOf}(\text{x}_\text{i}(\omega)) \geq \text{minLag}_{\text{j,i}} + \text{startOf}(\text{x}_\text{j}(\omega)); \qquad & \forall j \in J\ \forall i \in S_j \quad \forall \omega \in \Omega \end{aligned} \tag{6c}$$

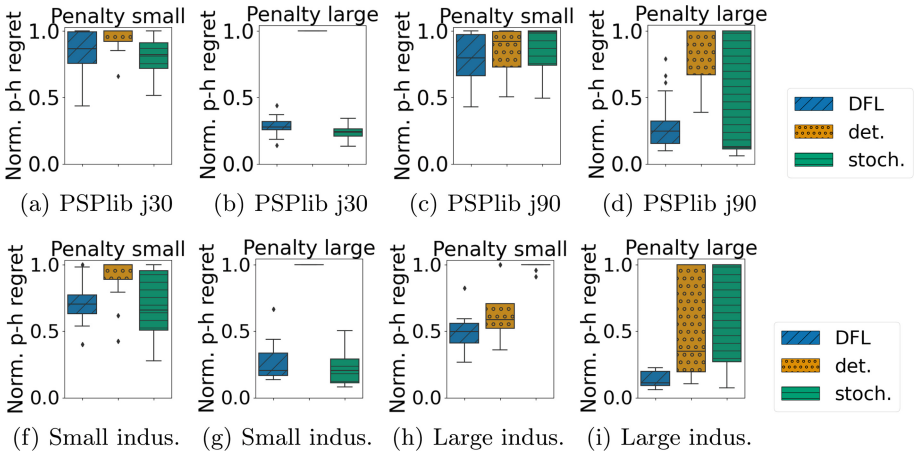
$$\sum_{j \in J} \text{Pulse}(\text{x}_\text{j}(\omega), \text{r}_{\text{r,j}}) \leq \text{b}_\text{r} \qquad \forall r \in R \quad \forall \omega \in \Omega \tag{6d}$$

$$x_j(\omega) : \text{IntervalVar}(\text{J}, \text{y}_\text{j}(\omega)) \qquad \forall j \in J \quad \forall \omega \in \Omega \tag{6e}$$

$$z_j \leq \text{startOf}(\text{x}_\text{j}(\omega)) \qquad \forall j \in J \quad \forall \omega \in \Omega \tag{6f}$$

5.3 Results

All experiments are done on a virtual server that uses an Intel(R) Xeon(R) Gold 6148 CPU with two 2.39 GHz processors, and 16.0 GB RAM. All CP models are solved with single thread IBM CP solver [4]. The runtime limits are set per



(a) PSPlib j30 (b) PSPlib j30 (c) PSPlib j90 (d) PSPlib j90

(f) Small indus. (g) Small indus. (h) Large indus. (i) Large indus.

Fig. 5. Normalized post-hoc regret per instance set - penalty setting (smaller regret is better). The box spans from the 25th to the 75th percentile, visualizing the median and interquartile range.

problem size (max. 60 min.) and provided in the README of the repository [8], together with the tuned hyperparameters[2] for deterministic, DFL, and stochastic. Each boxplot in Figure 5 presents the results for the three methods on a single combination of instance set and penalty setting. The y-axis shows the distribution of the normalized post-hoc regret among the test instances of that specific set. We use $\rho = \frac{1}{size}$ (small), where $size$ indicates the number of tasks, and $\rho = 1$ (large). We tested the significance of the performance difference of the different algorithms for each setup (a-i) using a paired t-test with $\alpha = 0.05$, and the p-values are included in the repository [8].

On smaller instances with small penalties, both DFL and stochastic methods perform well, with no significant difference. For the large penalty, stochastic tends to outperform both deterministic and DFL methods significantly in PSPlib $j30$ instances. Notably, under $\rho = 1$ for $j30$ instances, no repairs were needed across all instances, emphasizing the robust performance of stochastic when the instances are small enough. For larger instances like PSPlib $j90$, DFL becomes better, even significantly for the small penalty. For the large penalty, there is still a subset of the instances for which stochastic finds very robust solutions that do not need repairs, but because for some of the instances the stochastic model performs much worse than DFL (which is also visible in Fig. 5d), we observe no significant difference between DFL and stochastic looking at all $j90$ instances. We see a somewhat similar pattern in the industrial instances, where again stochastic is most advantageous for the smaller instances and with a high penalty, although not significantly better than DFL. For the larger instances, stochastic performs even worse, especially for the small penalty and DFL is significantly better. We

[2] Such as the number of scenarios.

investigated optimality gaps of the outputs of the stochastic model and found that even with a time limit of three hours the gaps are on average approximately around 30%, with outliers of more than 90% (for the largest industry instances) which shows the scalability issue of stochastic programming.

6 Related Work

Previous studies [2,6,13] focused mainly on comparing prediction-focused versus DFL approaches for problems with uncertainty in (linear) objectives. The knapsack problem is the most prominent [5,12]. As far as we know, the set-up without feature data is not studied in earlier work [13]. However, the crux of our problem setting is that uncertain parameters occur in constraints, which can lead to infeasibilities. Hu et al. [10] were the first who introduced a post-hoc regret that penalizes for infeasibilities. The methods introduced in their work rely on specific conditions, such as being recursively and iteratively solvable [9,10]. This work applies a DFL approach [17] to a pure stochastic repairable scheduling problem, for which both the repairable scheduling setting and the context without features are novel application domains for DFL. It is important to highlight that before this study we did not know if DFL could work for repairable scheduling.

7 To Conclude, and Continue

This study explores a novel application of DFL to stochastic resource-constrained scheduling with repairs, where uncertainty is in the constraints, and the derivative is not smooth by itself. Results indicate that stochastic programming is dominant when it can find the optimal solution, most prominently when the penalty factor is high and the instances are small enough to find robust solutions that do not need reparation. In contrast, we have shown that DFL scales better and is a promising alternative to stochastic programming, even in this pure stochastic scheduling setup. Furthermore, we highlight the potential of DFL because of its flexibility across various settings with different repair strategies, providing a distinct advantage over stochastic programming, in which modeling the exact repair functions is not always possible. We hypothesize that in a setting with features related to stochastic processing times the benefits of DFL for stochastic scheduling are further enhanced, such as shown in earlier research with uncertainty in a (linear) objective [13]. Further interesting directions are investigating alternative gradient estimators or reinforcement learning-inspired algorithms.

Acknowledgments. We acknowledge Mattia Silvestri and Michele Lombardi whose valuable insights improved the quality of this work. We are grateful that Léon Planken was available for questions related to the simulator. This work is supported by the AI4b.io program, a collaboration between TU Delft and dsm-firmenich, and is fully funded by dsm-firmenich and the RVO (Rijksdienst voor Ondernemend Nederland).

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

References

1. Ben-Tal, A., Ghaoui, L.E., Nemirovski, A.: Robust Optimization, 1st edn. Princeton University Press, Princeton (2009)
2. Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.P., Bach, F.: Learning with differentiable pertubed optimizers. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in neural information processing systems 2020, vol. 33, pp. 9508–9519. The MIT Press (2020)
3. Bertsimas, D., Kallus, N.: From predictive to prescriptive analytics. Manage. Sci. **66**(3), 1025–1044 (2019)
4. Cplex, IBM ILOG: V12. 1: User's manual for cplex. International Business Machines Corporation **46**(53), 157 (2009)
5. Demirović, E., et al.: An investigation into prediction + optimisation for the Knapsack problem. In: Rousseau, L.-M., Stergiou, K. (eds.) CPAIOR 2019. LNCS, vol. 11494, pp. 241–257. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19212-9_16
6. Elmachtoub, A., Grigas, P.: Smart "predict, then optimize". Manage. Sci. **68**(1), 9–26 (2022)
7. Glynn, P.W.: Likelihood ratio gradient estimation for stochastic systems. Commun. ACM **33**(10), 75–84 (1990). https://doi.org/10.1145/84537.84552
8. van den Houten, K.: Learning from scenarios for repairable stochastic scheduling (2023). https://github.com/kimvandenhouten/Learning-From-Scenarios-for-Repairable-Stochastic-Scheduling
9. Hu, X., Lee, J.C.H., Lee, J.H.M.: Branch and learn with post-hoc correction for predict+optimize with unknown parameters in constraints. In: Cire, A.A. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. LNCS, vol. 13884, pp. 264–280. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33271-5_18
10. Hu, X., Lee, J.C.H., Lee, J.H.M.: Predict+optimize for packing and covering LPs with unknown parameters in constraints. arXiv **2209.03668** (2022)
11. Kolisch, R., Sprecher, A.: PSPLIB - a project scheduling problem library. Eur. J. Oper. Res. **96**, 205–216 (1996)
12. Mandi, J., Demirović, E., Stuckey, P., Guns, T.: Smart predict-and-optimize for hard combinatorial optimization problems. In: Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI-20 (2020)
13. Mandi, J., et al.: Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. arXiv **2307.13565** (2023)
14. Matloff, N.: Introduction to Discrete-Event Simulation and the SimPy Language (2008)
15. Mohamed, S., Rosca, M., Figurnov, M., Mnih, A.: Monte Carlo gradient estimation in machine learning. J. Mach. Learn. Res. **21**(1), 5183–5244 (2020)
16. Ruszczyński, A., Shapiro, A.: Stochastic Programming, Handbook in Operations Research and Management Science. publisher (2003)
17. Silvestri, M., et al.: Score function gradient estimation to widen the applicability of decision-focused learning. arXiv **2307.05213** (2023)
18. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction, 2nd edn. The MIT Press, Cambridge (2018)
19. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**, 229–256 (1992)