UNIFY: A UNIFIED POLICY DESIGNING FRAMEWORK FOR SOLVING CONSTRAINED OPTIMIZATION PROBLEMS WITH MACHINE LEARNING

Mattia Silvestri, Allegra De Filippo, Michele Lombardi, Michela Milano
presented by Francesco Demelas

Introduction

- Decomposition of the policy in two stages:
  - an unconstrained Machine Learning Model
  - a C.O. problem

  The interface between the two is a new set of "virtual" parameters.
- Can generalize several existing approaches, thus extending their applicability, as:
  - Decision Focused Learning
  - Constrained Reinforced Learning
  - Algorithm Configuration
  - Stochastic Optimization
- Practical Problems:
  - Energy Management System Problem.
  - Set Multi-Cover with Stochastic Coverage Requirements Problem.

  For both, we assume to have implicit and explicit knowledge of the form:
  - historical data or simulators
  - an objective function
  - problem constraints

The Unify Framework - Introduction

Let $x \in X$ a vector of observable information and $z \in C(x) \subseteq \mathbb{R}^d$ a decision vector.
We aim defining a (constrained) policy function

$$\begin{aligned} \pi : & X \times \Theta \to \mathbb{R}^d \\ & (x, \theta) \to z \in C(x) \end{aligned}$$

where θ are the parameters that we want learn in such a way that π provides feasible (and possibly high-quality solutions).

The Unify Framework - Training and Inference

We can chose a θ using the

Training Problem

$$\theta^* \in \arg\min_{\theta \in \Theta} \mathbb{E}_{(x^+,x)\sim p} \left[f(x^+, x, \pi(x, \theta)) \right]$$

where:

- $f(x^+, x, z)$ is a function with real values, returning the cost of decisions z when x is observed and uncertainty unfold with outcome x^+ (w.l.o.g. $x^+ \in X$).
- p is the training distribution, that is the distribution approximated over the training set of simulators.

Once we have an optimal parameters vector θ^* we can take a decision on an unseen situation:

Inference Problem

$$x \to z = \pi(x; \theta^*)$$

Note

The training problem presented so far is **hard** to solve, since π will typically be trained on a large dataset and it is expected to return *always a feasible solution* from a space *possibly lacking a fixed structure*.
Anyway the training equation can be made easier by decomposing the policy into a *traditional* Machine Learning model and a *traditional* Combinatorial Optimization problem.
In formulas we consider a policy of the form

$$\pi(x, \theta) = g(x, h(x, \theta))$$

where:

- h is the ML model that provides as output a vector of **virtual parameters** $y = h(x, \theta)$.
- $g(x, y) \equiv \arg\min_{z \in \tilde{C}(x,y)} \tilde{f}(x, y, z)$ is an optimization problem defined with **virtual cost** $\tilde{f}$ and **virtual feasible set** $\tilde{C}$.

Reformulating the training problem

Now we can write

Training Problem

$$\theta^* \in \arg\min_{\theta} \mathbb{E}_{(x^+,x)\sim p}\left[f(x^+, x, z)\right]$$

with

$$z = g(x, y) = \arg \min_{z \in \tilde{C}(x,y)} \tilde{f}(x, y, z)$$

and $y = h(x, \theta)$

Note:

- If h is differentiable we can using the Sub Gradient Descent:
  - **forward pass**: evaluate h and then compute z.
  - **backward pass**: fix z and differentiate h w.r.t. θ.
- g can be built on the fly adjusting the size of the decision vector z as needed.
- Choosing y is a design decision as $\tilde{C}$ and $\tilde{f}$ depends on it. This means that g may differ from the original optimization problem and we can, for example, introduce:
  - penalty rewards

Sequential Formulation

Sequential Training Problem

$$\theta^* \in \arg\min_{\theta} \mathbb{E}_{\tau\sim p}[\sum_{k=1}^{eoh} \gamma^k f(x_{k+1}, x_k, z_k)]$$

$$z^k = g(x_k, y_k) = \arg \min_{z\in\tilde{C}(x_k,y_k)} \tilde{f}(x_k, y_k, z_k)$$

$$y_k = h(x_k, \theta)$$

where:

- τ is the trajectory of $\{(x_k, y_k, z_k)\}_{k=1}^{eoh}$
- eoh is the end oh the horizon and we should have:
  - $gamma = 1$ if $eoh < +\infty$
  - $0 < gamma < 1$ if $eoh = +\infty$

Sequential Formulation

The sequential formulation could be interpreted as defining a RL problem where h play the role of the conventional RL policy and g is at training time part of the environment and at inference time part of the policy that began $g(x, h(x, \theta^*))$.

Relation to Other Approaches

- Decision Focused Learning
- Constrained RL
- Algorithm Configuration
- Stochastic Optimization

Decision Focused Learning

Description:
Train estimator for parameters of optimization models, while explicitly accounting for the impact that estimation inaccuracy has on the decisions. DFL craft customized loss functions, which can improve convergence and solution quality for the training process.

Relation:

- All DFL approaches lack virtual parameters: the ML model is expected to estimate part of the future state, i.e. y is a prediction for a portion of x^+ .
- $\tilde{f}$ always corresponds to the original decision problem cost, and in few cases $C(x, y) = C$.
- UNIFY more flexible than DFL, however, it does not strictly generalize DFL, due to our use of the true cost f.

It should be possible to adapt DFL losses in UNIFY , and conversely some ideas (e.g. virtual parameters) could be adapted to DFL.

Constrained RL

Description: Learn within the same ML model how to satisfy the constraints and how to maximize the reward.

1. Reward shaping: constraint violation is modeled as a negative reward. Large penalty enforce constraint satisfaction, but can cause numerical instabilities and lead to poorly optimized policies.

2. After a gradient update, a projection adjusts the weights so that the decision vector becomes feasible This approach is more numerically stable, but also more computationally expensive.

Relation:

- In RL infeasible decisions may still occur on unseen examples. The second class of methods enforces constraint satisfaction via a projection in decision space (rather than in weight space). Guarantee constraint satisfaction, but it can still lead to lower rewards.
- The first class of approaches can be seen as approximately solving the unreformulated training problem. The second class can be assimilated to the sequential training problem, except that the ML output is already a decision vector (i.e. $y \in Z$), and that $\tilde{f}(x, y, z)$ is always the Euclidean distance.

Algorithm Configuration

Description:

- Adjusting the parameters of a given algorithm so as to optimize its behavior on a target distribution.
- The Dynamic Algorithm Configuration (DAC) approach, adjust the parameter values as the target algorithm progresses.

  DAC works by casting the configuration problem as a Contextual MDP and using RL to obtain a policy.

Relation:

- Traditional Algorithm Configuration methods can be interpreted as solving the reformulated training problem, while DAC as solving the sequential formulation.
- The vector y corresponds to algorithm parameters , rather than model parameters ,they do not take advantage of the flexibility offered by virtual parameters (1 exeption).

Stochastic Optimization

Description: Aims at improving robustness in single- ormulti-stage decision-making problems. Most stochastic optimization algorithms rely on Monte Carlo methods to approximate expected values and assess constraint satisfaction. The Sample Average Approximation (SAA) has long been a staple of the field, combined with Benders decomposition to address two-stage decision problems. Multi-stage decision problems have been often approximated as a sequence of 2-stage problems.

Relation:

- The stochastic optimization algorithms discussed here can be interpreted as addressing reformulated or sequential training problem in the case of AMSAA, without making use of a ML model, i.e. with $h(x, \theta) = x$.
- So, there is no training step and solving a new instance requires approximating the expected value from scratch, making the process quite expensive.

UNIFY suggests how by introducing a ML model the computational cost of sampling can be paid in a single offline phase, making inference much more efficient.

Practical Problems

- **Energy Management System**: Based on actual energy prices, and forecasts on the availability of DERs and on consumption, the EMS decides:
  1. how much energy should be produced;
  2. which generators should be used;
  3. whether the surplus of energy should be stored or sold to the energy market.
- **Set Multi-Cover**: Given a universe N containing n elements and a collection of sets over N , the Set Multi-cover Problem requires finding a minimum size sub-collection of the sets such that coverage requirements for each element are satisfied. The sets may represent products that need to be manufactured together, while the coverage requirements represent product demands

Energy Management System

$$\min \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{k=1}^{n} \sum_{g \in G} c_g^k x_{g,\omega}^k$$

$$s.t. \tilde{L}^k = \sum_{g \in G} x_{g,\omega}^k \quad \forall \omega \in \Omega, \forall k = 1, \cdots, n$$

$$0 \leq \gamma_k + \eta x_0^k \leq \Gamma \quad \forall \omega \in \Omega, \forall k = 1, \cdots, n$$

$$l_g \leq x_g^k \leq u_g \quad \forall \omega \in \Omega, \forall k = 1, \cdots, n$$

$$\gamma_\omega^{k+1} = \gamma_\omega^k + \eta x_{0,\omega}^k \quad \forall \omega \in \Omega, \forall k = 1, \cdots, n-1$$

$$x_{1,\omega}^{k+1} = \hat{R}_k + \xi_{R,\omega}^k \quad \forall \omega \in \Omega, \forall k = 1, \cdots, n$$

$$\tilde{L}_\omega^{k+1} = \hat{L}_k + y_k + \xi_{L,\omega}^k \quad \forall \omega \in \Omega, \forall k = 1, \cdots, n$$

Set Multi-Cover (Deterministic Version)

$$\min \sum_{j \in J} c_j x_j$$

$$\sum_{j \in J} a_{i,j} x_j \geq d_i \quad \forall i \in I$$

$$x_j \geq 0$$

$$x_j \in \mathbb{Z}$$

$$a_{i,j} \in \{0, 1\}$$

where I is the universe J is the collection of all possible sets, a is the availability matrix, d_i the requirement for the $i-$th element of the universe, c_j the cost of the $j-$th set and x the vector of all decision variables.

Set Multi-Cover (Stochastic Version)

$$\min \sum_{j \in J} c_j x_j + \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{i \in I} w_{i,\omega} s_{i,\omega}$$

$$\sum_{j \in J} a_{i,j} x_j \geq d_{i,\omega}(1 - z_{i,\omega}) \quad \forall i \in I$$

$$z_{i,\omega} \Rightarrow s_{i,\omega} \geq d_{i,\omega} - \sum_{j \in J} a_{i,j} x_j \qquad x_j \geq 0$$

$$z_{i,\omega} \in [0, 1]$$

$$s_{i,\omega} \geq 0$$

$$x, z \in \mathbb{Z}$$

where $\omega \in \Omega$ are the sampled scenarios, w the penalty vector, z a vector of indicator variables that allows the violation of requirement constraints and s is a vector of slack variables that keep track of the not satisfied demands.

Numerical Results

Use UNIFY to:

1. perform tuning of virtual parameters, similarly to Algorithm Configuration
2. enforce constraint satisfaction in Reinforcement Learning
3. obtain results analogous to DFL in a scenario where existing approaches are not applicable
4. improve decision robustness without relying on the Sample Average Approximation

Model Parameter Tuning

Practical Problem: EMS

Algorithm for Comparison: TUNING modification. Deep Reinforcement Learning as a black-box tool to find the optimal c st of the EMS described above.

Use of UNIFY:

- UNIFY - SINGLE - STEP: the ML model h provides the virtual costs for all the stages, namely $y = c^{st}\ \{1, ...96\}$.
- UNIFY - SEQUENTIAL : the model output at each step k is $y^k = c_k^{st}$

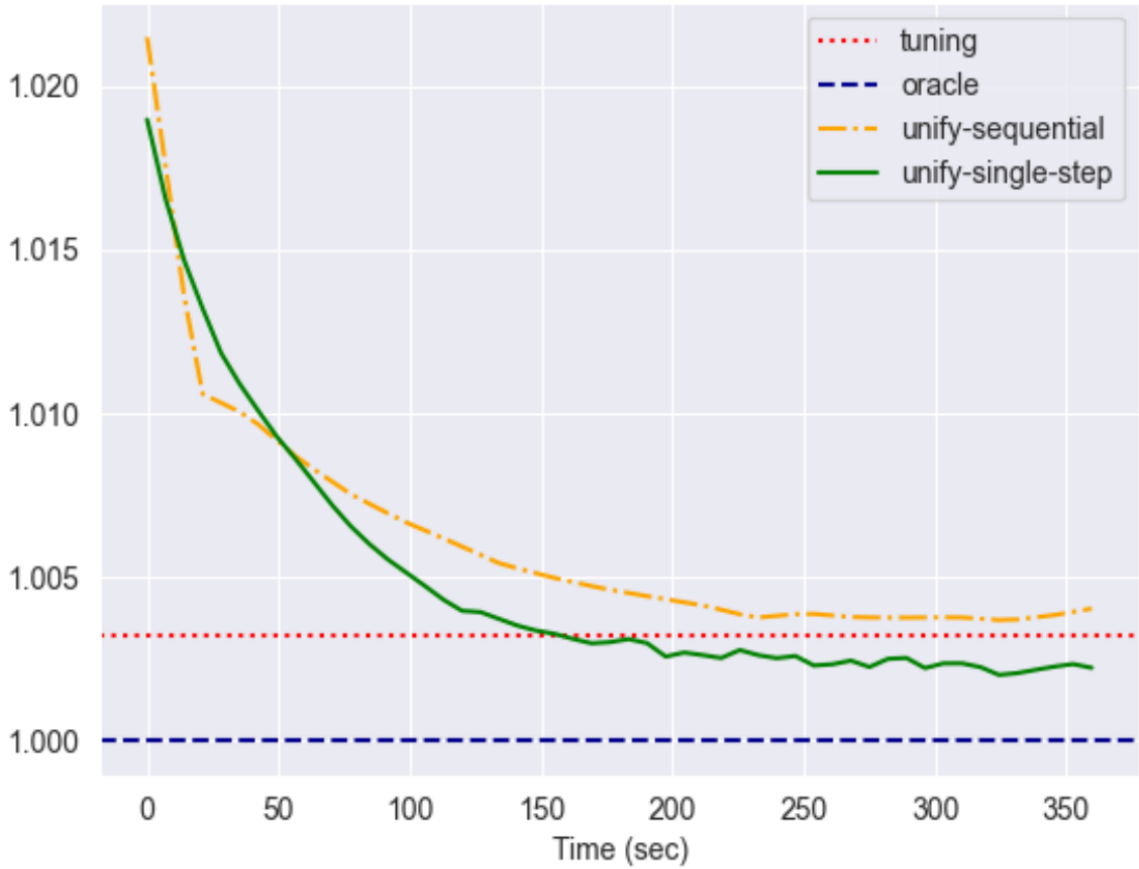

Figure: Optimality gap of the state-of-the-art TUNING approach and the UNIFY methods w.r.t. the computational time.

Constrained RL

Practical Problem: EMS

Algorithm for Comparison:
- end-to-end DRL algorithm to provide a solution, by learning constraints satisfaction only from the reward signal and refer to it as RL
- Projection-based DRL algorithms (e.g. Safety Layer) provide an alternative to full end-to-end methods when dealing with constraints: we have experimented with SAFETY - LAYER .

Use of UNIFY: in UNIFY - SEQUENTIAL the RL agent is indirectly guiding a problem-specific solver, which has the same guarantees in terms of feasibility, but it is arguably a simpler task.

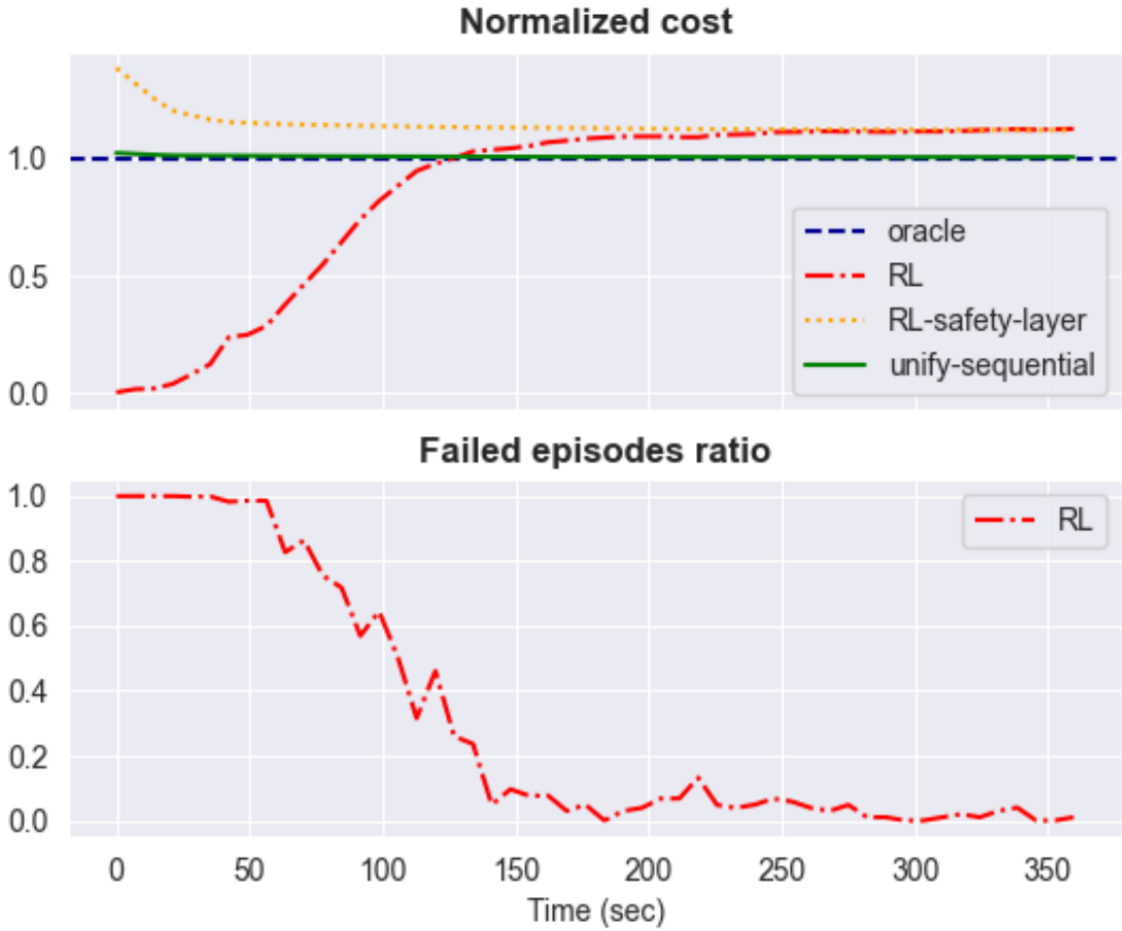

Figure: In this figure we show how demanding constraints satisfaction to the downstream solver greatly improves a full end-to-end RL method and SAFETY - LAYER.

UNIFY - SEQUENTIAL quickly converges as well, and it also improves the previous methods of a non-negligible gap. These experimental results demonstrate that Reinforcement Learning can benefit from a policy

Generalization of DFL

Practical Problem: Set Multi-Cover.

Algorithm for Comparison: predict-then-optimize .Train a ML model to accurately predict the rates of the underlying Poisson distribution that generates the stochastic coverage requirements, and then we use the predicted rates as the requirements in the Set Multi-cover optimization model.

Use of UNIFY: RL policy predicts the coverage requirements, $y = \{d_j\}_{j \in N}$ that are then plugged into the optimization model, and the overall policy is trained to minimize the solution cost.

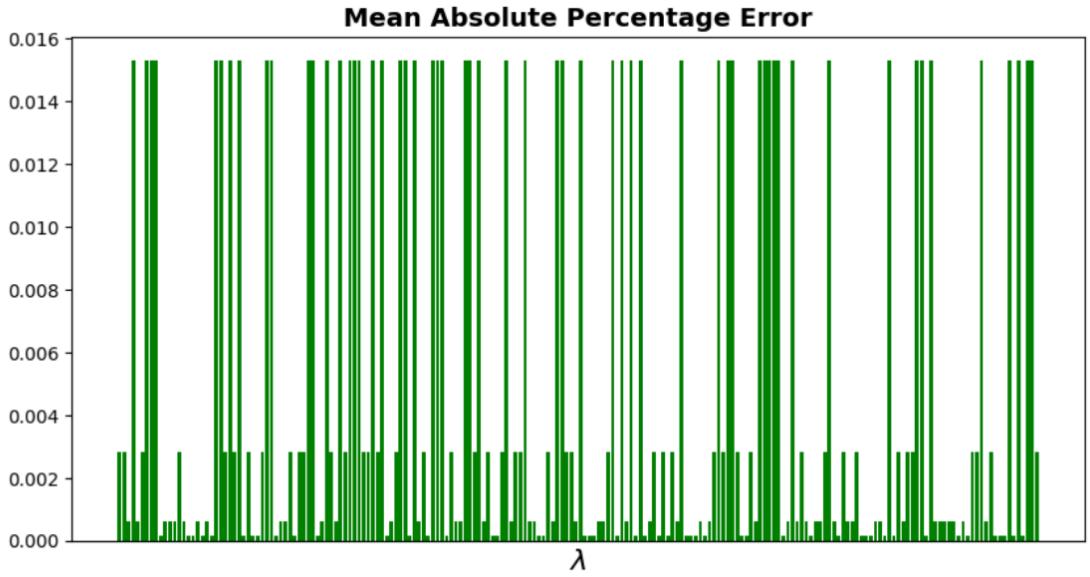

Figure: Here we report the predicted λ MAPE of the Machine Learning model employed in the predict-then-optimize approach.

The Mean Absolute Percentage Error (MAPE) for each of the rate $\lambda_i \ \forall i \in I$ is low, proving that the ML model is accurate.
The true task loss (solution cost) is far from optimal. the UNIFY implementation that is trained to directly minimize the task loss greatly outperforms the predict-then-optimize approach and it is considerably closer to the optimal cost.

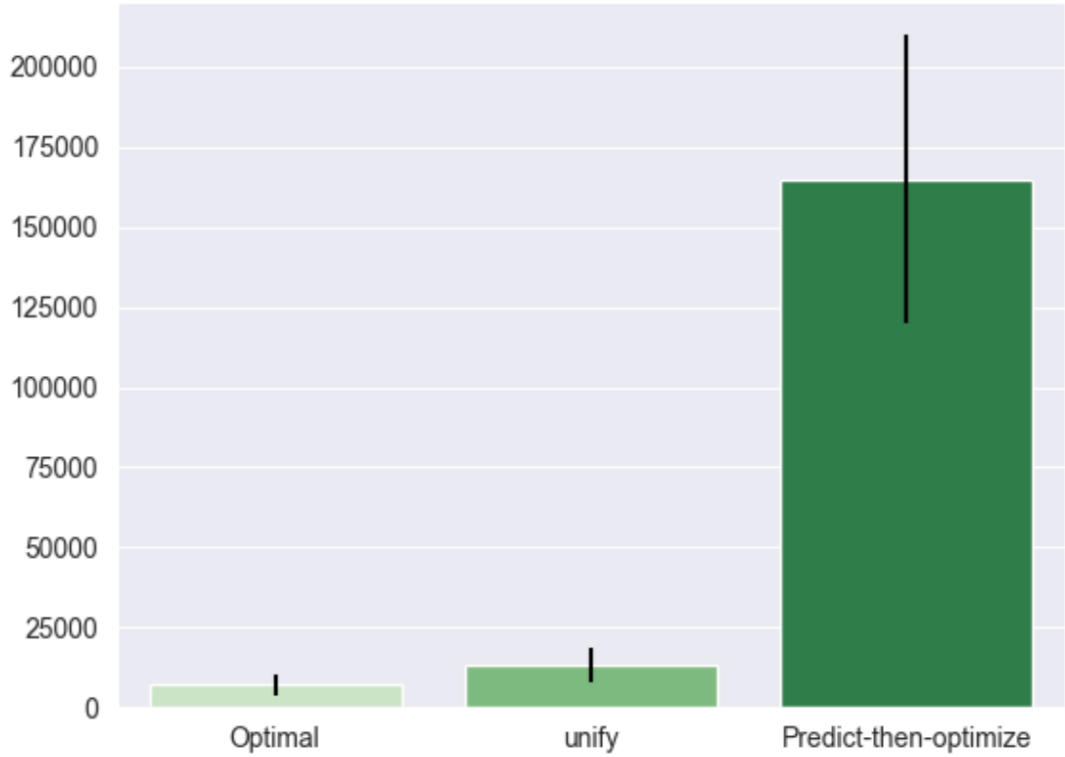

Figure: This figure reports the mean cost and standard deviation for the UNIFY and the Predict-then-optimize approaches compared to the optimal values.

We can conclude that UNIFY is a valid alternative to traditional Decision Focused Learning approaches, when those cannot be easily applied.

Stochastic Optimization

Practical Problem: Set Multi-cover

Algorithm for Comparison:
- SAA method that computes the optimal solution on a fixed set of instances (referred to as training instances).
- More robust baseline method that relies on the predict-then-optimize framework: ML model estimate the parametrization of the probability distribution then exploit the model predictions to generate samples.

Use of UNIFY: RL policy predicts the demands and the overall policy is trained to minimize the task loss, the same as described in the previous paragraph; notably, the latter approach does not rely on sampling. It is worth highlighting that the comparison favors the SAA method, since it is designed by assuming exact knowledge of the type of probability distribution; the UNIFY implementation makes no such assumption.

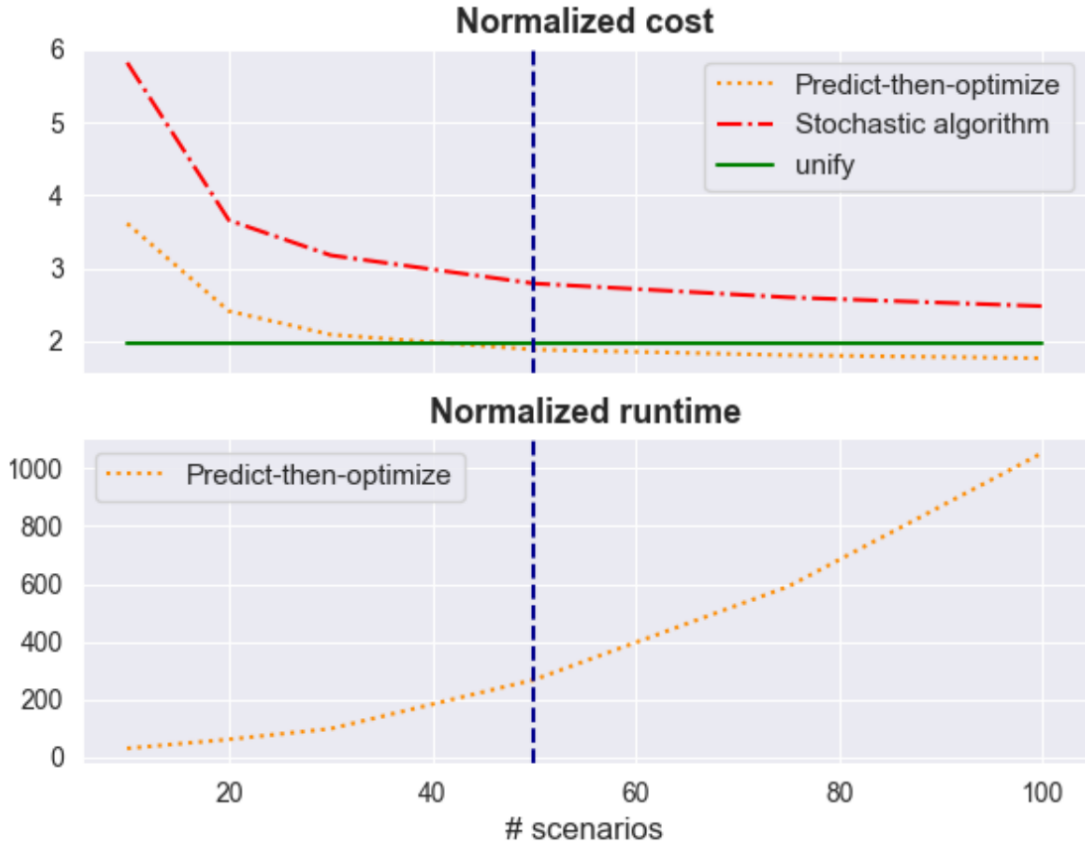

Figure: Optimality gap of the three methods on the Set Multi-cover problem with stochastic demands and the solution time of the predict-then-optimize approach w.r.t. the number of scenarios.

Both the simple SAA algorithm and predict- then-optimize approaches benefit from increasing the number of sampled scenarios.
UNIFY does not depend on the number of sampled scenarios, because it directly predicts the coverage values that are then plugged into the optimization model.
Predict-then-optimize surpasses UNIFY only when at least ~ 50 scenarios are used and, to obtain better results, predict-then-optimize requires more than 200 times the computation of UNIFY .
A smart implementation of UNIFY is a cheaper alternative to a SAA method for improving the robustness of the solver when facing stochastic optimization problems.