EURO Journal on Transportation and Logistics 9 (2020) 100023

Contents lists available at ScienceDirect

EURO Journal on Transportation and Logistics

journal homepage: www.journals.elsevier.com/euro-journal-on-transportation-and-logistics

Learning to handle parameter perturbations in Combinatorial Optimization: An application to facility location

Andrea Lodi [a,*], Luca Mossina [b], Emmanuel Rachelson [b]

[a] CERC, Polytechnique Montréal, Canada
[b] ISAE-SUPAERO, Université de Toulouse, France

ARTICLE INFO

Keywords:
Mathematical programming
Machine learning
Recurrent problems

ABSTRACT

We present an approach to couple the resolution of Combinatorial Optimization problems with methods from Machine Learning. Specifically, our study is framed in the context where a reference discrete optimization problem is given and there exist data for many variations of such reference problem (historical or simulated) along with their optimal solution. Those variations can be originated by disruption but this is not necessarily the case. We study how one can exploit these to make predictions about an unseen new variation of the reference instance.

The methodology is composed by two steps. We demonstrate how a classifier can be built from these data to determine whether the solution to the reference problem still applies to a perturbed instance. In case the reference solution is only partially applicable, we build a regressor indicating the magnitude of the expected change, and conversely how much of it can be kept for the perturbed instance. This insight, derived from *a priori* information, is expressed via an additional constraint in the original mathematical programming formulation.

We present the methodology through an application to the classical facility location problem and we provide an empirical evaluation and discuss the benefits, drawbacks and perspectives of such an approach.

Although it cannot be used in a black-box manner, i.e., it has to be adapted to the specific application at hand, we believe that the approach developed here is general and explores a new perspective on the exploitation of past experience in Combinatorial Optimization.

1. Introduction

Solving Combinatorial Optimization problems can be highly challenging. Many problems have been shown to be computationally hard to solve (Garey and Johnson, 1990), that is, no polynomial-time algorithm exists. Despite this intrinsic complexity, the state-of-the-art modeling and solving solutions can handle real-world instances via exact methods, heuristics or a mix of the two.

Our rationale goes as follows: given a combinatorial problem, we assume the existence of a reference problem *instance* P_{ref}, derived from the description of a system in its nominal state. This reference can be modeled via Mixed Integer Programming (MIP) and solved to its optimum x^*_{ref} via state-of-the-art solvers based on branch-and-bound (BB) methods like CPLEX (IBM, 2018). If a modification of the reference parameters occurs, the problem needs to be solved again. We operate as if we had to respond to this parameter perturbation (also referred to as *disruption*) under a tight optimization time budget, aiming at accelerating the descent (for a minimization problem) towards the optimum. Given the modifications that have occurred in the past or that can be simulated *a priori*, one can build a dataset of resolutions. Through the example of the Facility Location Problem, we study how one can extract information from this dataset to facilitate the resolution of future perturbed instances. We cast this problem as a Supervised Learning problem to predict whether a modification will affect and to what extent the reference optimal solution. According to this prediction, additional constraints are added to the MIP formulation of the perturbed instance. In order to evaluate the approach, the two formulations are fed to the solver and the performances are compared. Although a few steps of its implementation are specialized to the Facility Location problem, we believe our framework provides a general viewpoint to the use of data collected from the resolution of variations of the same problem. We find convenient and effective to present the methodology through the well-known and flexible Facility Location perspective (for example, to exploit clean data generation) but our aim is not to provide a *standalone* heuristic algorithm for the problem.

* Corresponding author.
E-mail addresses: andrea.lodi@polymtl.ca (A. Lodi), luca.mossina@isae-supaero.fr (L. Mossina), emmanuel.rachelson@isae-supaero.fr (E. Rachelson).

https://doi.org/10.1016/j.ejtl.2020.100023
Received 8 August 2020; Received in revised form 24 October 2020; Accepted 27 October 2020

Section 2 reviews the related literature on applying Machine Learning to Combinatorial Optimization problems. Section 3 provides a description of the Facility Location Problem and Section 4 introduces how the prediction problem can be formulated mathematically and linked to existing Machine Learning approaches. Experiments and numerical results are presented and discussed in Section 5. Section 6 discusses the advantages and drawbacks of the proposed method and presents future research directions.

2. Machine Learning for Combinatorial Optimization

The remarkable results achieved in recent years by Machine Learning (ML), for example in the field of image recognition (Krizhevsky et al., 2012) or Reinforcement Learning (Mnih et al., 2015), sparked a lot of interest in the Operations Research community. Some results have emerged in applications such as guiding the branching process in branch-and-bound optimization algorithms, in the form of node exploration ordering (He et al., 2014) and approximating performant branching rules (Alvarez et al., 2017; Khalil et al., 2016; Lodi and Zarpellon, 2017). In Di Liberto et al. (2016) one can find algorithmic ideas on how to handle the many existing heuristics already developed for branch-and-bound methods.

The traveling salesman problem has been addressed via an ad-hoc neural network architecture in a Reinforcement Learning context (Bello et al., 2017; Khalil et al., 2017; Deudon et al., 2018), aiming at learning to build a solution end to end.

Another field of application is the interaction between existing optimization solvers and *a priori* knowledge on the problem. For instance, Kruber et al. (2017) find an approach to automatically handle MIP problem decompositions, by detecting if and what reformulation to apply within a dedicated software. On the same topic, Basso et al. (2018) bring evidence to why such an approach can be carried out, showing empirically that the role of an expert needed when using decomposition methods can be, at least in part, automated via learning algorithms. In Bonami et al. (2018), the authors present a method to select the best resolution method for a Mixed Integer Quadratic Programming problem from the different algorithms offered within the CPLEX framework.

Machine Learning has proven useful in producing a description of the optimal solution of a yet unseen instance. Fischetti and Fraccaro (2019) describe a real-world application with the problem of a wind turbine park layout. Minimizing the complex turbulence induced by the positioning of wind turbines generates a series of difficult combinatorial problems. Where different configurations would need to be evaluated at a considerable computational cost, the authors approximate the solution values to these optimizations via ML, thanks to a dataset built *a priori*. On a similar line lies the work of Larsen et al. (2018). When the problem of choosing the optimal load planning for containers on freight trains cannot be solved online because of time constraints and insufficient information (tactical level), a set of offline cases can be collected and used to get a description of the solution of a new instance, at an aggregate level. Such description provides meaningful insights to decision makers in a real-time context.

Strictly connected with our work, Xavier et al. (2019) consider a number of ML techniques to extract information from previously solved instances of Unit Commitment problems and leverage such pieces of information to improve the MIP performance when solving similar instances again and again.

The interested reader is referred to Bengio et al. (2018) for a recent survey on the subject.

We end the section by pointing out that our approach is inherently apart of the line of work of Bello et al. (2017) and also slightly different from Xavier et al. (2019). On the one side, we assume the existence of a reference instance and we are using ML (through associated data) to speed up the resolution of its variations occurred because of parameter disruptions. In other words, we are not developing an end-to-end heuristic to be applied to general instances of a Combinatorial Optimization model, which is what Bello et al. (2017) do for the TSP. On the other side, although Xavier et al. (2019) is also leveraging a group of solutions to find common ingredients that could speed up resolution of a new one, again they do not assume a reference instance as well as disruptions in its parameters.

3. Capacitated facility location problems

Our application of choice will be that of the Single-Source Capacitated Facility Location Problem (SSCFLP). Given a group of customers and a list of potential facilities that can serve them, one must open a certain number of facilities and assign a unique facility to each of the customers. The goal of the problem is to satisfy the demand of the customers while minimizing the associated operational and start-up costs. The facilities are constrained in capacity, that is, the quantity of total service each can provide is limited. When solving this problem one determines which facilities must be opened and which customers they will serve.

3.1. Mathematical programming formulation

The SSCFLP can be written as a Binary Integer Program, with decision variables y_j, $x_{ij} \in \{0,1\}$, indicating respectively whether facility j is activated (also referred to as *open*) and whether customer i is served by j.

$$\underset{x,y}{\text{minimize}} \quad \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{j\in J} f_j y_j \tag{1}$$

$$\text{subject to} \quad \sum_{i\in I} d_i x_{ij} \le s_j y_j \quad \forall j \in J \tag{2}$$

$$\sum_{j\in J} x_{ij} = 1 \quad \forall i \in I \tag{3}$$

$$x_{ij} \in \{0,1\},\ y_j \in \{0,1\} \quad \forall i \in I, \forall j \in J \tag{4}$$

$$I = \{1,2,\ldots,N_C\}, J = \{1,2,\ldots,N_F\}, \tag{5}$$

where $i \in I$ are the customers and $j \in J$ the available facilities; $c_{ij} \ge 0$ is the cost of serving customer i from j; $f_j \ge 0$ is the fixed cost for opening facility j; $d_i \ge 0$ is the demand of customer i; $s_j \ge 0$ is the capacity of facility j; N_F and N_C are the number of facilities and of customers, respectively.

The objective function (1) quantifies the total cost of a given assignment, composed of the fixed start-up costs $\sum_{j\in J} f_j y_j$ and operational costs $\sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij}$. Constraints (2) ensure that the total demand of the customers assigned to j does not exceed its capacity s_j. Constraints (3) ensure that each customer is served by exactly one facility. For convenience, we write an instance as $P = (\boldsymbol{c}, \boldsymbol{f}, \boldsymbol{d}, \boldsymbol{s})$.

In the various forms of the Facility Location Problem (FLP), the binary variables associated with facility activation make the problem computationally challenging.[1] This is the reason why we specialize our methodology to learn (about) them, see Section 4. For a comprehensive treatment on exact formulations of the FLP, we refer the reader to the work of Klose and Drexl (2005). Beyond mathematical programming and exact methods, heuristics for the FLP have received a fair amount of attention (Cornuéjols et al., 1983; Guastaroba and Speranza, 2012, 2014).

[1] Even the uncapacitated version of the FLP, for instance, has been proven to be NP-hard (Cornuéjols et al., 1983).

3.2. Perturbations in a reference instance

We assume the existence of a reference instance $P_{ref} = (\boldsymbol{c}_{ref}, \boldsymbol{f}_{ref}, \boldsymbol{d}_{ref}, \boldsymbol{s}_{ref})$ where the parameters are considered to be in their nominal state. This problem has $\{\boldsymbol{x}^*_{ref}, \boldsymbol{y}^*_{ref}\}$ as its reference solution. In case a disruption in the system occurs, for example an anomaly in the capacity, we wish to learn whether the reference solution, or part of it, is still applicable without running the full optimization. To that end, we wish to predict what parts of the solution need to be changed by providing hints to the solver, via an additional constraint. We shall write a perturbed instance $P' = (\boldsymbol{c}', \boldsymbol{f}', \boldsymbol{d}', \boldsymbol{s}')$, which we see as a variation $P' = P_{ref} + \Delta P$, where a disruption ΔP affected one or more of P_{ref}'s parameters. In practice, one would proceed as in the following example. The given (simple) P_{ref} with capacities $\boldsymbol{s}_{ref} = (5000, 5000, 5000)$ is affected by a disruption and the derived P' has capacities $\boldsymbol{s}' = (5000, 5088, 4304)$. The ML algorithm will take $\Delta P = \{0, 88, -696\}$ as input and predict how $\{\boldsymbol{x}'^*, \boldsymbol{y}'^*\}$ could differ from $\{\boldsymbol{x}^*_{ref}, \boldsymbol{y}^*_{ref}\}$, before running the optimization for P' (see Section 4). In our case, the hint to the solver will specify how many (if any at all) of the facilities active in P_{ref} will be still active, without specifying which of the three.

3.3. FLP and framework generality

The FLP basic structure can apply to different domains. Consider, for instance, the airport system of a region. If a runway at a major airport has problems or needs maintenance, one could think of this as a disruption. One could then want to learn and predict how to react to the reduction of the capacity of that runway, how much of the traffic would need to be delayed, rescheduled or rerouted or, in general, how to be guided to manage the disruption.

Overall, we could say that, on the one hand, our framework is more suitable for operational problems than for strategic or tactical ones, to which the FLP class mostly belongs. On the other hand, FLP features, practically, a two-level structure, where deciding on a specific subset of the variables (i.e., the location ones in FLP) heavily simplifies the problem. Thus, learning on the variations of this special subset over a reference solution is especially relevant and our framework can be designed accordingly.

Indeed, this is the main design step that has to be considered to apply the proposed framework to a Combinatorial Optimization problem: deciding on which variables it is most relevant to learn so as to be able to constrain the nominal formulation accordingly (see constraint (6) in the next section).

4. Learning constraints

In this Section, we formalize the search for a constraint that will be used to accelerate the resolution of SSCFLP instances facing disruptions. We formulate this as a statistical learning problem and describe how we couple it with MIP resolution.

4.1. Learning to bound the changes to a reference solution

Given the information gained from past resolutions or simulated data, we investigate the possibility of predicting the number of facilities we expect to change. We want to predict that "only a proportion $(1-\gamma)$, say 0.15, of the facilities needs replanning", and impose that as a constraint. Indeed, it is well known in the case of MIPs with binary variables (Fischetti and Lodi, 2003) that it is easy to express the neighborhood of a feasible solution by a linear constraint and the resulting MIP is generally way easier to solve than the original problem. In addition, for FLP it is also well known that the critical choice is associated with the facilities to open while the assignment of clients to facilities is easier to deal with.

Then, the problem boils down to predicting the proportion of facilities opened in $\boldsymbol{y}^*_{ref}$ to be kept open in $\boldsymbol{y}'^*$, the optimal solution of the perturbed instance P'. This estimation is a scalar value $\widehat{\gamma}$ determined via a function $\Gamma(\Delta P)$, where $\Gamma : \mathbb{R} \to [0,1]$. The function depends on the disruption of the data of the original instance. Given the $\{(\Delta P_k, \gamma_k)\}_{1 \le k \le K}$ data from the K past resolutions, this is a regression problem, which we can tackle with any Statistical Learning method (Hastie et al., 2009).

The drawback of such an approach is the potential introduction of bias in the solution of P' if the true optimum of P' is far from $\{\boldsymbol{x}^*_{ref}, \boldsymbol{y}^*_{ref}\}$ and the ML fails to detect it or if it cuts the optimum from the feasible domain. In fact, in case the parameters $(\boldsymbol{c}', \boldsymbol{f}', \boldsymbol{d}', \boldsymbol{s}')$ generating P' are very different from P_{ref}, a good model would impose no additional constraint, thus falling back to the full problem.

The information on $\widehat{\gamma} = \Gamma(\Delta P)$ can be modeled through the linear constraint (6), thus yielding the new MIP

$$\begin{aligned} &\underset{x,y}{\text{minimize}} \quad \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{j \in J} f_j y_j \\ &\text{subject to} \quad \text{all previous constraints and} \\ &\sum_{r \in R_{ref}} y_r \ge \widehat{\gamma}\, N_{ref}, \end{aligned} \tag{6}$$

where $R_{ref} = \{j \in J \,|\, y^*_{ref\,j} = 1\}$ is the set of indices of the facilities opened in the reference solution, $\widehat{\gamma} \in [0,1]$ is the parameter predicted via ML and $N_{ref} = |R_{ref}|$ is the number of facilities activated in the reference problem.

We do not attempt to identify directly which of the facilities need to change or stay as in $\{\boldsymbol{x}^*_{ref}, \boldsymbol{y}^*_{ref}\}$, but rather to restrict the number of facilities that can be changed. If $\widehat{\gamma} \approx 0$, the disruption data conveys no information as to the new optimum and all the facility allocations should be left to the solver. In that case, it is likely that the variation on P_{ref} is so strong that prior information is of no help and we need to solve the perturbed instance as a full problem in its original formulation. Conversely, if $\widehat{\gamma} \approx 1$, we conclude that all of the reference facilities are to be left activated, without excluding further facilities from being activated.

4.2. Structuring the prediction problem

The overall problem of predicting $\widehat{\gamma} = \Gamma(\Delta P)$ is a regression problem. Previously solved instances P_k provide values $(\Delta P_k, \gamma_k)$ to assemble a training set. Then, most ML methods will search for Γ by minimizing a loss function defined as an expected value of individual losses over the training set, such as the least squares. Such methods can thus be sensitive to the training samples' distribution and might overfit to the majority value. In the specific case of predicting the proportion of facilities to keep open, the distribution of observed values for samples $(\Delta P_k, \gamma_k)$ is strongly affected by the fact that in many cases, *all* the facilities in the reference solution should remain open (for details, see Section 5.1). To compensate for this imbalance, our predictor's architecture features two levels.

First, we fit a binary classifier to indicate whether the open facilities in $\{\boldsymbol{x}^*_{ref}, \boldsymbol{y}^*_{ref}\}$ should all be kept open. If this is the case, then $\widehat{\gamma} := 1$ without any further computation because we assume that only the capacities have been perturbed/disrupted. Second, if we cannot assert that all the reference active facilities need to be left active, we call a specific regression function. This $\Gamma(\Delta P)$ function is trained on all the data points except those for which $\gamma_k = 1$. Then, given a perturbed problem P', we predict how much of the reference solution has to be changed, adding that as constraint (6) to the original problem. Fig. 1 summarizes the $\widehat{\gamma}$ estimation process and the resolution of a perturbed instance P'.

5. Experimental evaluation

Given the reference instance P_{ref} and an instance P' derived by perturbing P_{ref}, our aim is to provide a good solution to P' within a small time budget.

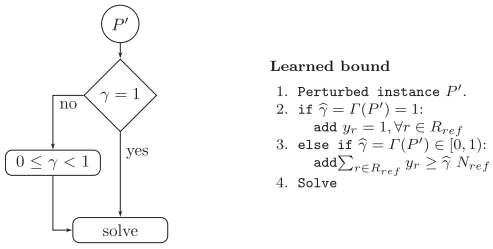


Fig. 1. Learning to bound the SSCFLP.

Our P_{ref} is taken from the SSCFLP instances of Guastaroba and Speranza (2014).[2] We test our method on three different reference cases, respectively $capa_1$, $capb_1$ and $capc_1$. For conciseness, we refer to these as A,B,C, and their corresponding MIP models as $P^A_{ref}, P^B_{ref}, P^C_{ref}$. In each case, we have $N_F = 100$ facilities available that share the same capacity $s^A = s^A_1 = s^A_2 = \ldots = s^A_{100}$.

5.1. Generation of training data

For each of the three instances, we produce three datasets of sizes $N_A = N_B = N_C = 10000$. Given P_{ref}, we generate our training data by perturbing the capacities of the facilities. We randomly perturb between 5% and 50% of the facilities as follows.

1. Pick uniformly at random between 5% and 50% of the $N_F = 100$ facilities.
2. For each facility j picked at Step 5.1, apply a Gaussian noise to its capacity: $s_j \leftarrow s_j + \mathcal{N}(0, \sigma^2)$, with $\sigma = 0.2 \times s_j$.
3. All the remaining data is left unchanged. This yields ΔP_k.
4. Solve the perturbed instance $P_k = P_{ref} + \Delta P_k$ with a MIP solver and get the number Y_k of facilities that were in x^*_{ref} and are also in x^*_k.
5. Add the new point $(\Delta P_k, \gamma_k = Y_k / N_F)$ to the dataset.

After generating the data, we can observe the distribution of Y, the number of original open facilities still open in the optimal solution of the randomized instances. At their nominal values, the optimal solutions for A, B and C have respectively 7, 11 and 11 facilities open. For instance, given the k-th point in our generated data from case A, if $Y_k = 4$, this means that after the perturbation of the capacities, out of the seven facilities open in the reference optimal solution, four were included in the optimal solution of the derived problem P_k.

Fig. 2 presents three different scenarios. With instance A we see that on average less than half of the 7 original facilities are kept open, that is, the perturbations affect the new optimal solutions. In the case of B, for most of the data points the original reference solution is still optimal after the perturbations, as seen in the right-most bar of Fig. 2b. Case C is intermediate, with at least eight facilities out of eleven still open after the variations of the capacities.

5.2. Learning the prediction model

As mentioned in Section 4.2, we need a binary classifier and a regressor to account for sample distribution imbalance. The number of data for binary classification depends on the shape of the generated data, that is, on how many extreme cases we observe (rightmost column in histograms of Fig. 2).

We split the training data into classification and regression data as follows. In case A, we have 717 data points of value 7, corresponding to the observations where all the facilities open in the reference solution are to be left open in the perturbed instances. Thus, we set these 717 points aside and randomly sampled an equal amount of points among the other points, yielding a learning data set of $N_{Classification}A = 1434$. The remaining 9283 points form the regression learning data.

This process was repeated for all the three cases, yielding the distributions reported in Table 1.

In accordance to standard practice in ML (Hastie et al., 2009), at learning time the data set was randomly split into a *training* and a *test* partition, namely 2/3 training and 1/3 test. The predictive models were fit on the training data but evaluated on the test data.

5.2.1. Comparison of ML algorithms: classification

For the binary classification task, we tested a set of binary classifiers among which Extremely Randomized Trees (Extra Trees) (Geurts et al., 2006), Neural Network classifier (Goodfellow et al., 2016), Logistic Regression classification and Naive Bayes classification (Hastie et al., 2009).

As it can be seen across Table 2, Extra Trees emerged as the best performing in terms of accuracy and false positives errors on the test data, that is, over the total of the prediction, how many were predicted as false positives. Here, false positives are cases where the reference solution's open facilities should *not* be all kept open in the perturbed instance but the classifier prescribes otherwise. This could introduce bias, making this metric important for our task.

5.2.2. Comparison of ML algorithms: regression

For the regression task we selected Extremely Randomized Trees out of a pool of models comprising Multiple Linear Regression, Extremely Randomized Trees and a Multi-layer Perceptron artificial Neural Network.

Table 3 reports the Mean Squared Error (MSE) computed on the test partition. Extra Trees yielded the best performance of the tested methods. While Neural Networks have recently been the object of intense research and remarkable results, we think its poorer results can be explained by the limited amount of training data ($N \leq 10000$) and the highly nonlinear relations between features and response variable, for which more data could have been needed.

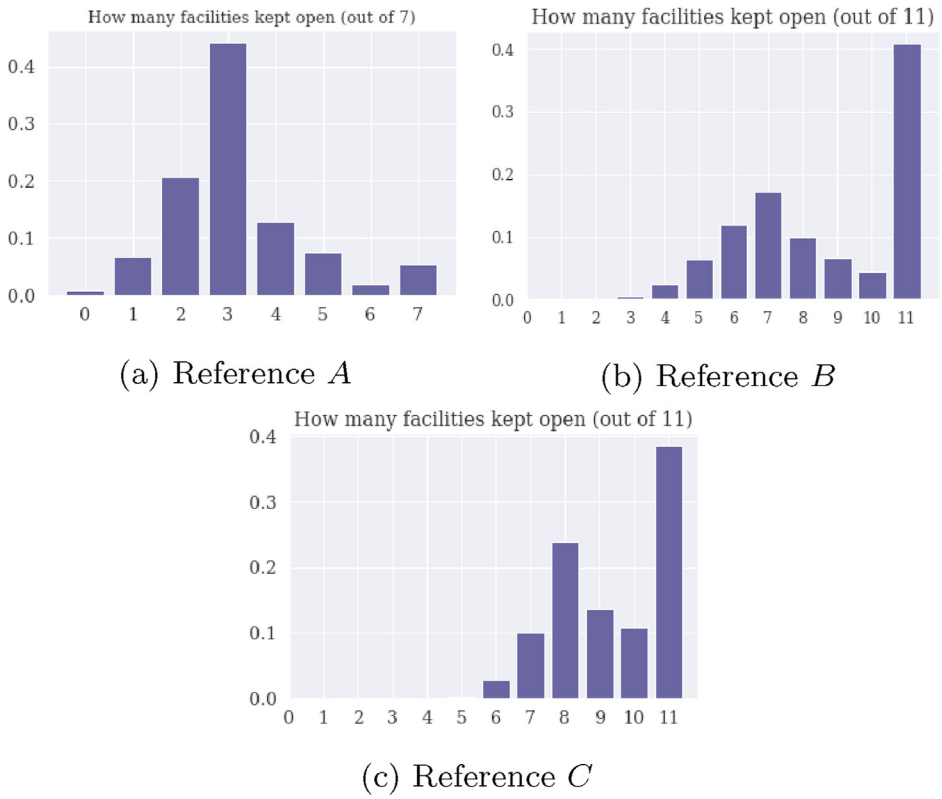
Fig. 3a, c and 3e, plot the predictions of Y (number of facilities kept open) versus the true values on the test data. The ideal case of perfect predictions $\widehat{Y} = Y$, is denoted by the red line. The green line is a simple linear regression of the form $\widehat{Y} = \widehat{m}Y + \widehat{b}$ that graphically renders the actual relation between Y and $\widehat{Y}$. Finally, Fig. 3b, d and 3f plot the distribution (as histograms) of the frequency of the absolute errors on the number of facilities kept open after perturbation. Such an error is highly distributed around 0, although some peaks occur with the predicted value slightly higher than the observed one.

We remark that the data are highly concentrated around some central values. The tails of the distribution are scarcely represented in our training data, and the learning function cannot generalize well, as one can see more prominently with instance A, where most of the data take values 2, 3 or 4.

5.2.3. Sensitivity analysis

A natural question for ML algorithms is the sensitivity of the learning results to the amount of data. To assess that, for each of the three reference instances, we repeated ten times the following procedure: we set apart 500 data points to be used as test data, that is, the data on which is tested the performance of the trained model; then, we trained four times the learning model (Extra Trees in both classification and regression) on, respectively, a subsample of 500, 1000, 2000 and 5000 data points. Due to the lack of data points (see Table 1), for the classification task in reference instance A, we run the same test as above but on a smaller scale: 100 test data points, and 100, 200 and 500. The four prediction models were tested on the same test data, to avoid introducing noise in this phase. The results of this experiment are reported in Fig. 4, where Fig. 4a, c, and 4e (resp. 4b, 4d and 4f) concern the classification (resp. regression) task and the 10 lines correspond to the 10 different tests.

[2] Data available at http://or-brescia.unibs.it/instances/instances_sscflp.



(a) Reference A (b) Reference B

(c) Reference C

Fig. 2. Number of facilities kept after reoptimization.

Table 1
Training Data: repartition between binary classification and regression.

Reference	Classification	Regression	Total
A	1434	9283	10000
B	8056	5965	10000
C	7686	6179	10000

Table 2
Comparison of binary classifiers.

Reference	Model	Accuracy	False Positive Errors
A	Extra Trees	0.756	0.106
	Neural Net	0.717	0.160
	Logistic	0.648	0.189
	Naive Bayes	0.641	0.191
B	Extra Trees	0.840	0.049
	Neural Net	0.857	0.083
	Naive Bayes	0.782	0.100
	Logistic	0.629	0.192
C	Extra Trees	0.864	0.033
	Neural Net	0.858	0.085
	Naive Bayes	0.776	0.104
	Logistic	0.616	0.199

Table 3
Comparison of regression models by Mean Squared Error.

Reference	Linear Regression	Extra Trees	Neural Net
A	0.998	0.883	0.975
B	2.087	1.780	2.099
C	1.114	0.750	1.097

The analysis of the results in Fig. 4 does not provide surprises: both the accuracy of the classification (the higher the better) and the MSE of the regression (the smaller the better) improve if more data are used for training. A certain variability can be observed among the 10 lines, which is also expected and it seems that the best results obtained with 5000 points (500 for the classification task for reference instance A) are comparable with those in Tables 2 and 3 that are obtained with roughly double of the data points. In other words, it would be relatively safe to halve the number of points for learning.

5.3. Optimization results

We report the performance of our method on the three references A, B and C, generating three batches D_A, D_B and D_C of 50 new instances, unseen during training. We apply the same procedure followed to generate the learning data (Section 5.1).

To measure the effect of the ML bound (see constraint (6)) with respect to resolution time, each instance in D_A (but also D_B and D_C) is optimized twice (with and without bound) with time limits $t_{lim} \in \{5, 10, 30, 60, 120\}$ seconds. At the end of the time-limited run, we record the objective value and the effective solving time (some instances are optimized before the limit is reached).

Our ML constraint (6) cannot make the solution infeasible, as the total number of active facilities is not bounded, but it could cut off the optimal solution. For instance, one could predict all of the reference facilities to be open ($\hat{\gamma} = 1$) while none of them should be. We run the optimization with 3600 seconds of time limit, to measure the difference in objective values after a long run, with and without the additional constraint.

5.3.1. Experimental setup

For each iteration of our experiments, we proceed as follows. A perturbed instance is generated as detailed in Section 5.1. The instance is then solved once for each time limit without the additional constraint. The objectives and effective times are recorded. The ML algorithm is run (see Section 4) and constraint (6) is introduced in the model's formulation. The constrained model is then solved once for each time limit. To

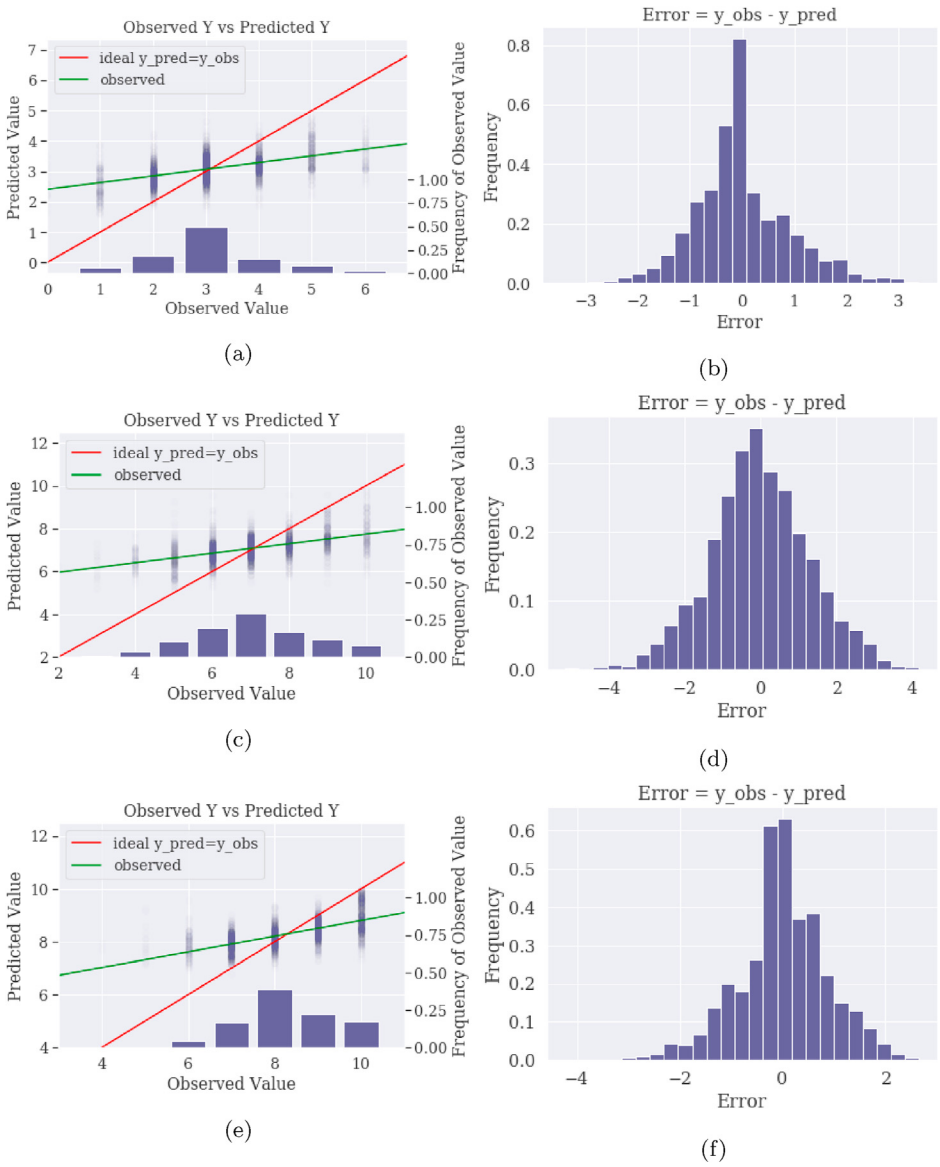


Fig. 3. References *A* (a,b), *B* (c,d) and *C* (e,f): Learning, regression.

verify that the optimal solution was preserved by the introduced bound constraint, we run both models with and without constraint (6) for a full hour to check if any remarkable deviation has been caused by the ML.

When using a highly-optimized MIP solver like CPLEX, it is sometimes hard to evaluate the impact of model modifications (imposed by the user) because the complex algorithmic components within the solver might mitigate the effect of those modifications. For this reason, it is common practice to deactivate, for testing purposes, some of those CPLEX algorithmic ingredients.

In our special case, we are interested in evaluating the effect of the ML-predicted constraint (6) as a way to fast dive toward good solutions of a modified SSCFLP instance with respect to the solution of a reference one. To achieve this goal, we have found useful to run CPLEX without the *presolve* feature and we first report the results of this configuration. We nonetheless performed all experiments and report the results with CPLEX *default*, i.e., with *presolve* activated. We show that the overall message provided by the experiments is in both cases very similar.

Finally, given the interest, especially in the academic community, of using non-commercial MIP solvers, we report in Appendix A the results obtained by replacing CPLEX with the most widely adopted of those solvers, namely SCIP (*default* version) (Gleixner et al., 2017).

5.3.2. *Experimental results*

Within 3600 seconds, most of the runs attain an optimality gap of the order of 0.05% or manage to reach optimality. The bias introduced by the learned bound is on average around 0.03% and at most of the order of 0.5%. That is, at the 3600 seconds time limit, the objective value of the ML-bounded run is on average 0.03% higher than the objective for the regular run, but never greater than 0.5%.

Fig. 5a, c and 5e compare the values of the objective functions of the constrained and unconstrained resolutions at the time limit. Each point represents the same perturbed, unseen, instance optimized twice: once with the ML bound and once without. The color coding helps determining the time limit imposed on this pair of runs. On the x axis one can see the objective value without ML, on the y axis the value with the ML bound. Points on the lower left corner correspond to runs with longer time limits, where the difference between the two approaches becomes more negligible. The points below the line are runs for which our method outperformed the original formulation.

In Fig. 5b, d and 5f, the same information is presented focusing on the temporal dimension. On the x axis is the effective resolution time, which can be inferior to the time limit. Along the y axis is the objective value at the time limit. The points aligned at the bottom of Fig. 5d and

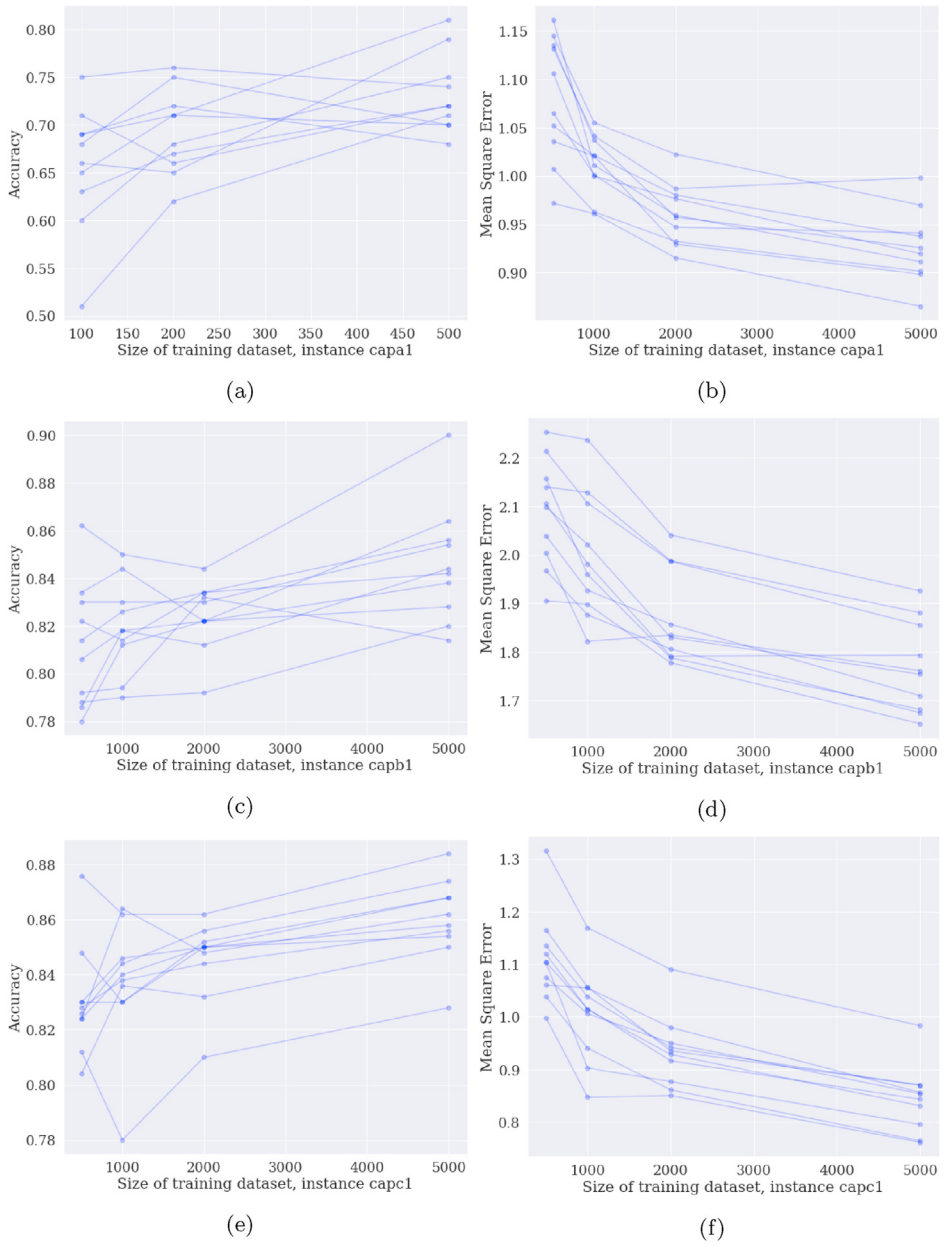


Fig. 4. References *A* (a,b), *B* (c,d) and *C* (e,f): sensitivity to data quantity.

f are those for which the optimal solution was found before the time limit expired. For short resolution times, the ML-constrained formulation yields better objective function values than the original formulation without cutting the optimal solution which is eventually found given more computational time. On average, the blue points representing the objective of the ML-bounded model lie below the red ones of the unbounded model. In case *C*, our method allows for superior performances than those obtained simply via the solver. In case *A* we are still on average improving the performances or at least keeping in line with the unbounded model. Here, the amplitude of the perturbations imply completely different solutions, making the transfer of information from the original solution to the new one difficult, i.e., very little remains the same. We remark however that, on average, our approach avoids cutting the optimal solution and thus prevents a sort of *negative* transfer.

As noted above, we operated with the *presolve* feature switched off. For completeness, we run all of the experiments a second time with the *presolve* feature activated (Fig. 6). The results reflect what was observed previously, although the presolve heuristics of CPLEX manage to reduce in part the effect of our approach in case *B* (Fig. 6 c and d).

The declared interest of our methodology is on applying it for a short computing time, i.e., in a recovery-mode fashion. This is the reason why, in the previous experiments, we concentrated on short time limits. Nevertheless, a natural question to ask is which is the effect of the constraint learned by ML on the overall computing time/quality of the solution when the standard time limit of 3600 is kept. In the attempt of answering such a question, we have computed the quantity

$$\Delta_{OBJ} = \frac{obj_{Free} - obj_{ML}}{obj_{Free}}\%,$$

where *obj* refers to the objective value at 3600 seconds of optimization. Whenever $\Delta_{OBJ} > 0$, the final solution of the ML-constrained instance has yielded a lower, hence better, solution value. Whenever this value is negative, it means that, *in the long term*, the ML constraint has introduced some bias (but not necessarily within the shorter time limits, i.e., in 5–120 s).

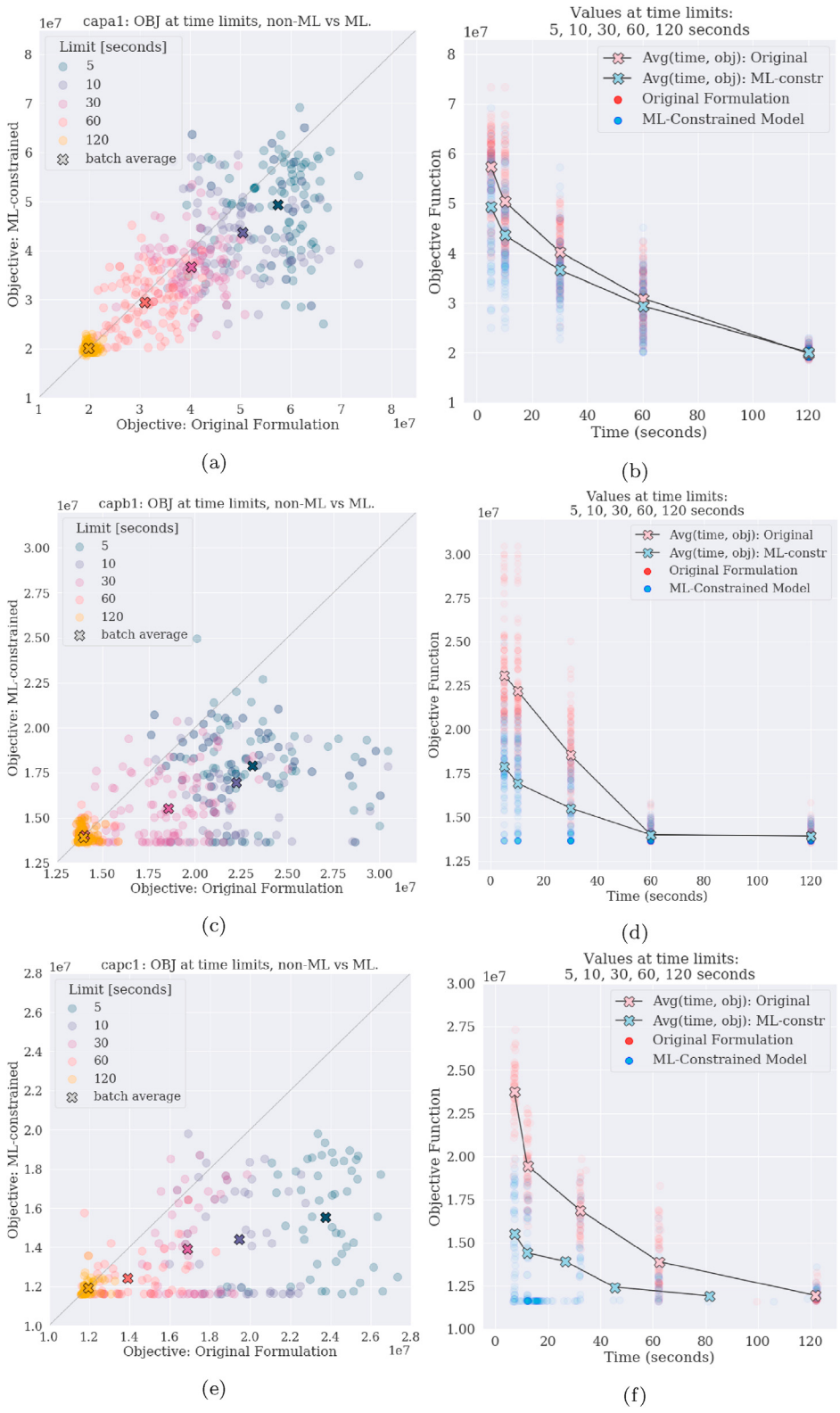


Fig. 5. References *A* (a,b), *B* (c,d) and *C* (e,f): optimization results.

This quantity alone, however, does not explain the whole picture. We also introduced the quantity

$$\Delta_{Time} = time_{Free} - time_{ML}$$

to account for the corresponding optimization time for the two cases, with and without the ML constraint. Indeed, in some cases, the optimization was completed before the limit of 3600 s. Hence, whenever $\Delta_{Time} > 0$, the ML-constrained optimization was completed Δ_{Time} seconds before the non constrained version. We point out that this could correspond to two cases: both instances were solved in less than 3600 seconds or the ML-constrained arrived at the optimum while the other, at 3600, still had not reached it. The inverse is true for negative values.

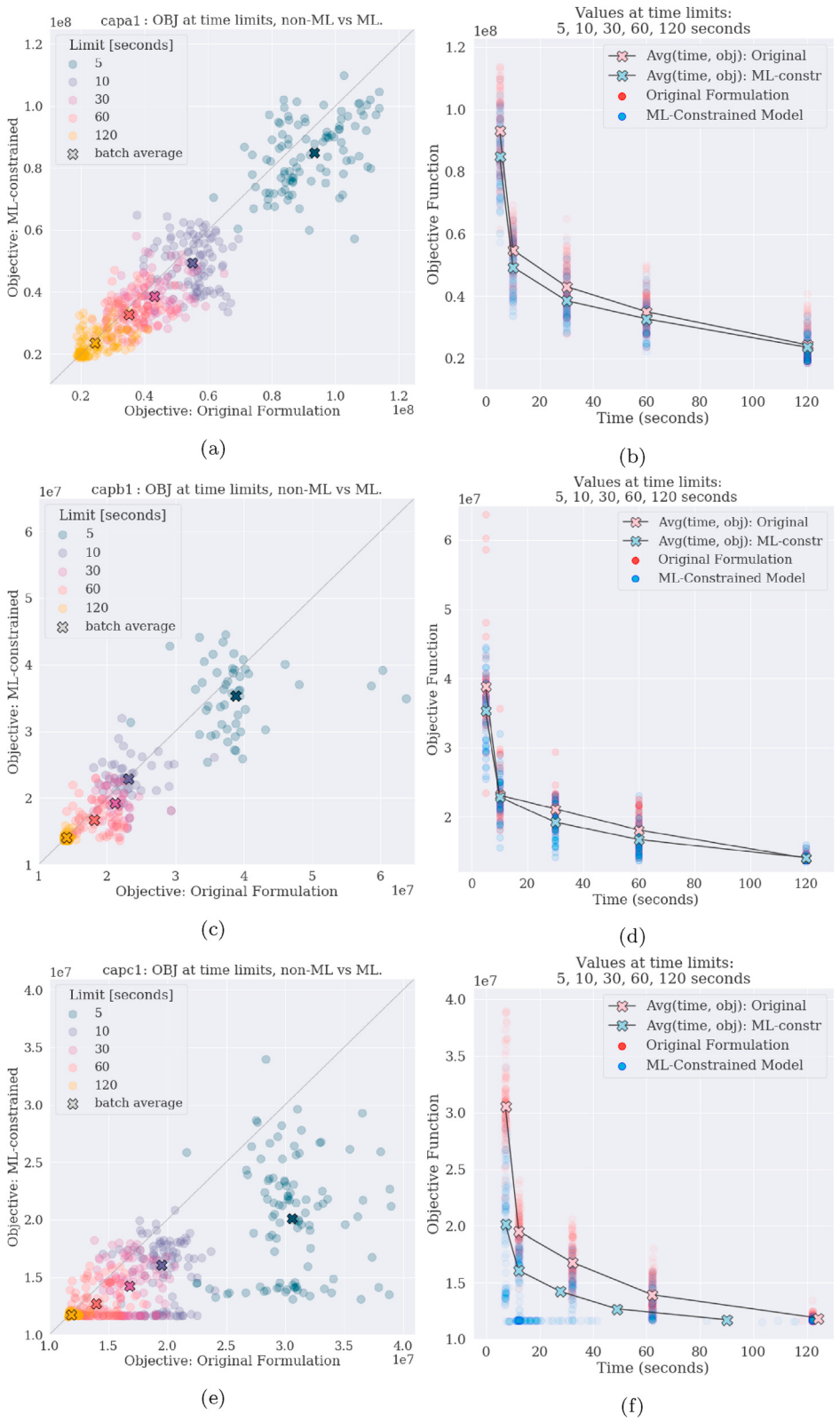


Fig. 6. References *A* (a,b), *B* (c,d) and *C* (e,f): optimization results with *presolve*.

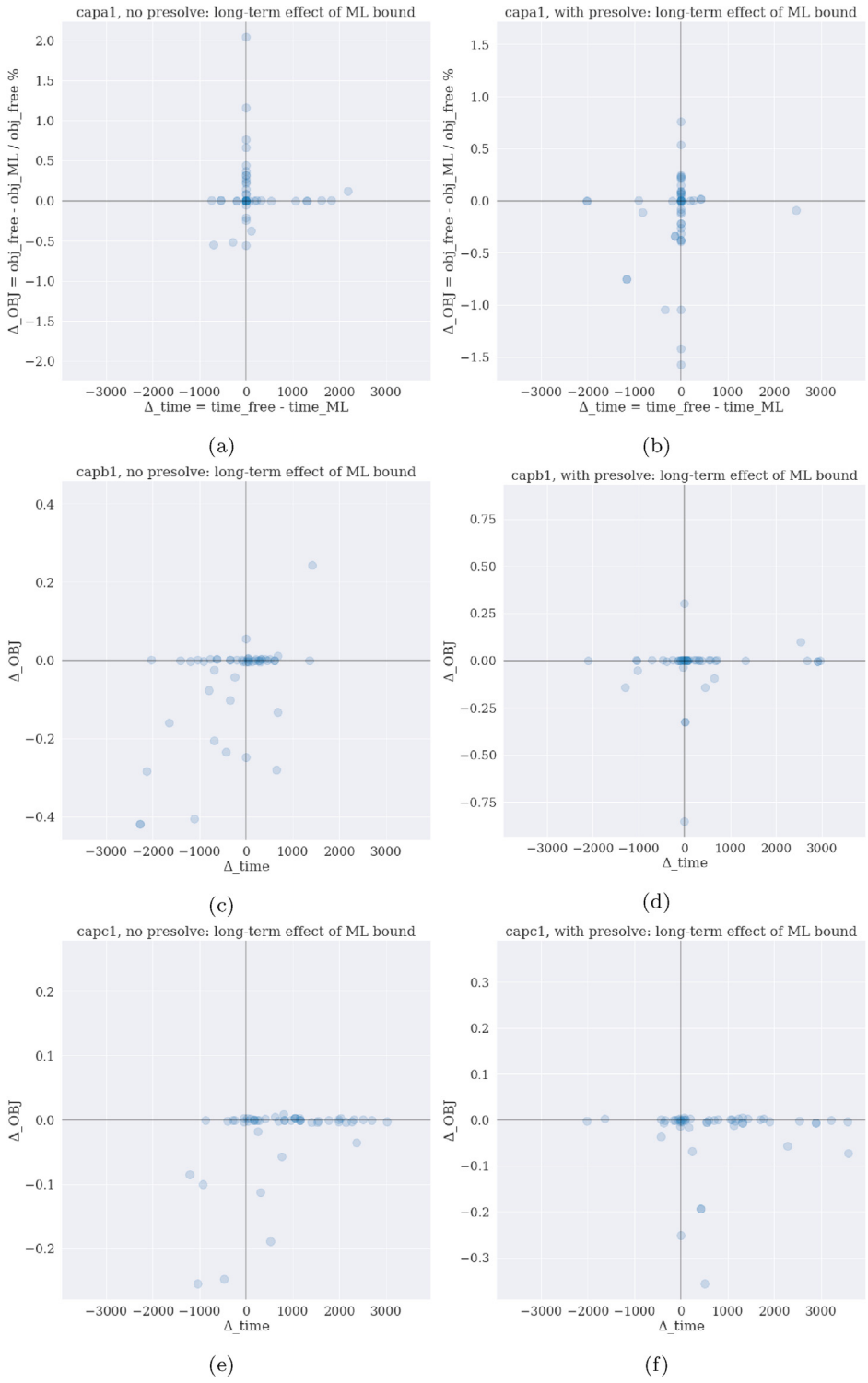


Fig. 7. References *A* (a,b), *B* (c,d) and *C* (e,f): left *without* presolve, right *with* presolve.

Table 4
Comparative results between CPLEX with "warm start" and the ML-constrained model.

Reference	Presolve	Time Limit (sec.s)				
		5	10	30	60	120
capa1	False	0.45	0.45	0.35	0.30	0.45
capa1	True	0.30	0.20	0.45	0.20	0.35
capb1	False	0.50	0.65	0.55	0.35	0.25
capb1	True	0.40	0.50	0.50	0.40	0.30
capc1	False	0.85	0.80	0.70	0.80	0.50
capc1	True	0.45	0.25	0.40	0.40	0.40

Fig. 7 reports the results of the experiment on our three datasets of 50 unseen instances each, the left (resp. right) plots being with CPLEX without (resp. with) presolve. Overall, the results indicate that 1) the bias introduced by the ML model is never too high (number and distance of the points below the horizontal axis), 2) there are a few cases in which not only the ML-constrained optimization is faster but also obtains a better solution (because of the standard solver reaching time limit, first quadrant), and 3) the prevalence with or without presolve is for points (instances) on the horizontal axis, i.e., with the same solution value but on the right-hand side of the vertical axis, i.e., in which the ML-constrained optimization is faster.

A final experiment compares our framework with CPLEX standalone fed by the reference solution as a "warm start". The experimental setting is as follows. For every reference instance A, B and C, we used 20 perturbed instances and solved them with CPLEX powered by the reference solution as a "warm start" and with the ML constraint (6) for all 5 time limits. We count the number of times solving the ML-constrained instance provided a better solution and Table 4 reports such a number normalized by 20. In other words, an entry 0.45 in Table 4 indicates that ML-constrained solution has been better in 45% of the cases.

The results in Table 4 show that using the reference solution as a "warm start" is certainly a good heuristic for solving the perturbed instances. Nevertheless, there is still a significant number of instances in which the ML-constrained model achieves a better solution (especially if the CPLEX presolve is deactivated). We note that the ML-constrained model could benefit from the "warm start" as well but we decided to not use it to assert the quality of the learned constraint itself. In other words, both the "warm start" and the ML constraint (6) can be combined.

6. Conclusions and perspectives

We have shown that the existing or simulated data of a recurring optimization problem can be exploited to gain insight into the optimization process. We have used these data to fit a binary classifier and a regressor. We predict whether a perturbed instance, derived from a reference one, will share all or a part of its optimal solution with the solution of the reference instance. This piece of information, translated into an additional constraint, is given to a solver and allows to dive faster, on average, towards a good solution. Moreover, we empirically illustrated that this additional constraint preserves the optimal solution and thus prevents negative transfer.

Building upon these results, we plan on extending the reach of our approach by experimenting with other, more operational families of problems, where, potentially, different types of constraints should be considered. This will be the way of fully demonstrating the wide applicability of the approach, especially in the context of online variations of a reference problem. In addition, the framework should be tested on problems in which a clear two-level structure is not straightforward to spot, so as to assert if some implicit information can be inferred. Finally, it would be interesting to extend our framework to a case in which there is more than one reference instance, so as to potentially develop a method that is a combination between the current one and that of Xavier et al. (2019).

Declaration of competing interest

No conflict of interest to be declared.

Acknowledgements

The authors wish to acknowledge the support of the Canada Excellence Research Chair in Data Science for Real-Time Decision-Making at Polytechnique Montréal and the ISAE-SUPAERO foundation. We are indebted to two anonymous referees for a careful reading and useful criticism that helped us to improve the positioning of the framework and its description.

Appendix A

Results with SCIP

The MIP solver SCIP, whose source code is accessible for academic purposes, has a considerable adoption in academic research, which is why it is interesting to test our approach in conjunction with such a solver. The experiments reported in Fig. 8 are the same as those in Section 5.3, with the exception of the time limits of 5 seconds; for such a short time limit, most of the runs did not yield a feasible solution and we did not report it. What was observed previously with CPLEX is true here as well, although the effect of constraint (6) is amplified. The additional bound constraint allows the solver to dive much faster towards a good solution. Even at the 120 seconds mark, the gap between the two objective values is still considerable.

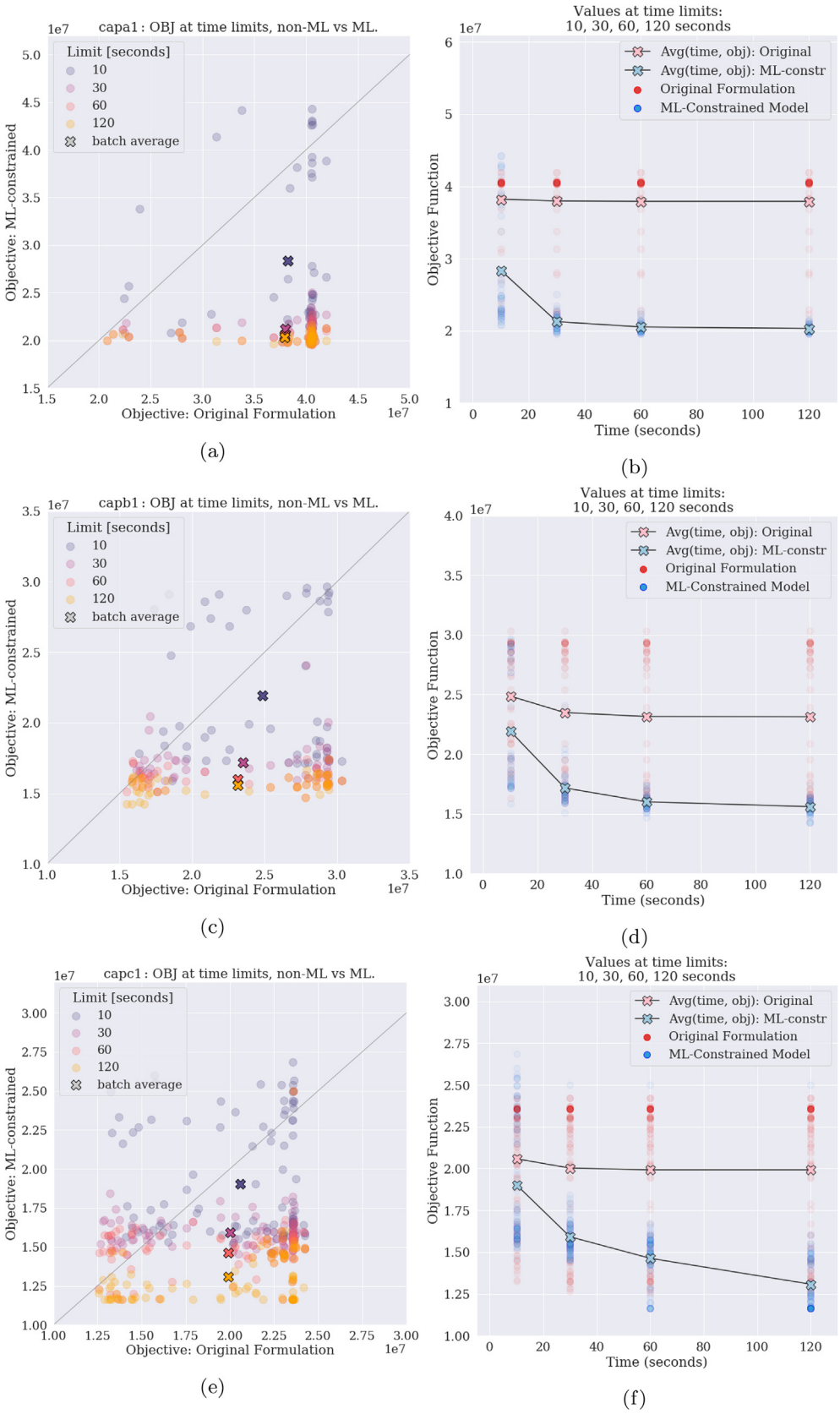


Fig. 8. References *A* (a, b), *B* (c,d) and *C* (e,f): optimization results with the SCIP solver.

References

Alvarez, A.M., Louveaux, Q., Wehenkel, L., 2017. A machine learning-based approximation of strong branching. Inf. J. Comput. 29 (1), 185–195.

Basso, S., Ceselli, A., Tettamanzi, A., 2018. Random sampling and machine learning to understand good decompositions. Ann. Oper. Res. https://doi.org/10.1007/s10479-018-3067-9. URL https://doi.org/10.1007/s10479-018-3067-9.

Bello, I., Zoph, B., Vasudevan, V., Le, Q.V., 2017. Neural optimizer search with reinforcement learning. In: International Conference on Machine Learning, pp. 459–468.

Bengio, Y., Lodi, A., Prouvost, A., 2018. Machine Learning for Combinatorial Optimization: a Methodological Tour D'horizon. Preprint arXiv:181106128.

Bonami, P., Lodi, A., Zarpellon, G., 2018. Learning a classification of mixed-integer quadratic programming problems. In: van Hoeve, W.J. (Ed.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research. Springer International Publishing, Cham, pp. 595–604.

Cornuéjols, G., Nemhauser, G.L., Wolsey, L.A., 1983. The Uncapacitated Facility Location Problem. Tech. Rep. Carnegie-mellon univ pittsburgh pa management sciences research group.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.M., 2018. Learning heuristics for the tsp by policy gradient. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research. Springer, pp. 170–181.

Di Liberto, G., Kadioglu, S., Leo, K., Malitsky, Y., 2016. Dash: dynamic approach for switching heuristics. Eur. J. Oper. Res. 248 (3), 943–953. https://doi.org/10.1016/j.ejor.2015.08.018. URL. http://www.sciencedirect.com/science/article/pii/S0377221715007559.

Fischetti, M., Fraccaro, M., 2019. Machine learning meets mathematical optimization to predict the optimal production of offshore wind parks. Comput. Oper. Res. 106, 289–297. https://doi.org/10.1016/j.cor.2018.04.006. ISSN 0305-0548. http://www.sciencedirect.com/science/article/pii/S0305054818300893.

Fischetti, M., Lodi, A., 2003. Local branching. Math. Program. 98, 23–47.

Garey, M.R., Johnson, D.S., 1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA.

Geurts, P., Ernst, D., Wehenkel, L., 2006. Extremely randomized trees. Mach. Learn. 63 (1), 3–42.

Gleixner, A., Eifler, L., Gally, T., Gamrath, G., Gemander, P., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schlösser, F., Serrano, F., Shinano, Y., Viernickel, J.M., Vigerske, S., Weninger, D., Witt, J.T., Witzig, J., 2017. The SCIP optimization suite 5.0. Technical report, Optimization Online, URL. http://www.optimization-online.org/DB_HTML/2017/12/6385.html.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. http://www.deeplearningbook.org.

Guastaroba, G., Speranza, M., 2014. A heuristic for bilp problems: the single source capacitated facility location problem. Eur. J. Oper. Res. 238 (2), 438–450.

Guastaroba, G., Speranza, M.G., 2012. Kernel search for the capacitated facility location problem. J. Heuristics 18 (6), 877–917. https://doi.org/10.1007/s10732-012-9212-8. URL https://doi.org/10.1007/s10732-012-9212-8.

Hastie, T., Tibshirani, R., Friedman, J., 2009. The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer Series in Statistics.

He, H., Daume III, H., Eisner, J.M., 2014. Learning to search in branch and bound algorithms. In: Advances in Neural Information Processing Systems, pp. 3293–3301.

IBM, 2018. IBM ILOG CPLEX optimizers 1280. URL. https://www.ibm.com/analytics/cplex-optimizer.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L., 2017. Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems, pp. 6348–6358.

Khalil, E.B., Le Bodic, P., Song, L., Nemhauser, G., Dilkina, B., 2016. Learning to branch in mixed integer programming. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence.

Klose, A., Drexl, A., 2005. Facility location models for distribution system design. Eur. J. Oper. Res. 162 (1), 4–29. https://doi.org/10.1016/j.ejor.2003.10.031. URL. http://www.sciencedirect.com/science/article/pii/S0377221703008191 (logistics: From Theory to Application).

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105.

Kruber, M., Lübbecke, M.E., Parmentier, A., 2017. Learning when to use a decomposition. In: Salvagnin, D., Lombardi, M. (Eds.), Integration of AI and OR Techniques in Constraint Programming. Springer International Publishing, Cham, pp. 202–210.

Larsen, E., Lachapelle, S., Bengio, Y., Frejinger, E., Lacoste-Julien, S., Lodi, A., 2018. Predicting Solution Summaries to Integer Linear Programs under Imperfect Information with Machine Learning. arXiv preprint arXiv:180711876.

Lodi, A., Zarpellon, G., 2017. On learning and branching: a survey. Top 25, 207–236.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529.

Xavier, A., Qiu, F., Ahmed, S., 2019. Learning to Solve Large-Scale Security-Constrained Unit Commitment Problems. Technical Report 1902.01697, arXiv.