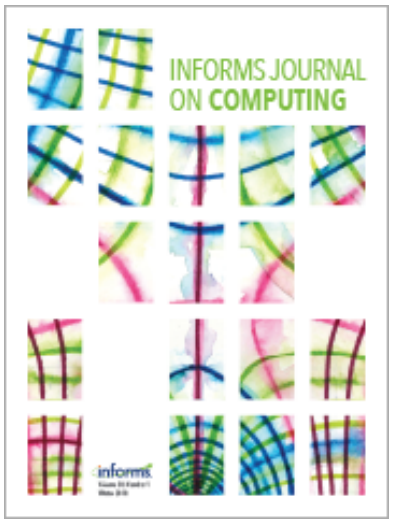


INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Learning to Solve Large-Scale Security-Constrained Unit Commitment Problems

Álinson S. Xavier, Feng Qiu, Shabbir Ahmed

To cite this article:
Álinson S. Xavier, Feng Qiu, Shabbir Ahmed (2021) Learning to Solve Large-Scale Security-Constrained Unit Commitment Problems. INFORMS Journal on Computing 33(2):739-756. https://doi.org/10.1287/ijoc.2020.0976

http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 33, No. 2, Spring 2021, pp. 739–756
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Learning to Solve Large-Scale Security-Constrained Unit Commitment Problems

Álinson S. Xavier,[a] Feng Qiu,[a] Shabbir Ahmed[b]

[a] Energy Systems Division, Argonne National Laboratory, Argonne, Illinois 60439; [b] School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

Contact: axavier@anl.gov, https://orcid.org/0000-0002-5022-9802 (ÁSX); fqiu@anl.gov (FQ); sahmed@isye.gatech.edu (SA)

Received: February 15, 2019
Revised: October 1, 2019; February 17, 2020
Accepted: March 7, 2020
Published Online in Articles in Advance: October 9, 2020

https://doi.org/10.1287/ijoc.2020.0976

Abstract. Security-constrained unit commitment (SCUC) is a fundamental problem in power systems and electricity markets. In practical settings, SCUC is repeatedly solved via mixed-integer linear programming (MIP), sometimes multiple times per day, with only minor changes in input data. In this work, we propose a number of machine learning techniques to effectively extract information from previously solved instances in order to significantly improve the computational performance of MIP solvers when solving similar instances in the future. Based on statistical data, we predict redundant constraints in the formulation, good initial feasible solutions, and affine subspaces where the optimal solution is likely to lie, leading to a significant reduction in problem size. Computational results on a diverse set of realistic and large-scale instances show that using the proposed techniques, SCUC can be solved on average 4.3 times faster with optimality guarantees and 10.2 times faster without optimality guarantees, with no observed reduction in solution quality. Out-of-distribution experiments provide evidence that the method is somewhat robust against data-set shift.

Summary of Contribution. The paper describes a novel computational method, based on a combination of mixed-integer linear programming (MILP) and machine learning (ML), to solve a challenging and fundamental optimization problem in the energy sector. The method advances the state-of-the-art, not only for this particular problem, but also, more generally, in solving discrete optimization problems via ML. We expect that the techniques presented can be readily used by practitioners in the energy sector and adapted, by researchers in other fields, to other challenging operations research problems that are solved routinely.

History: Accepted by Andrea Lodi, Area Editor for Design and Analysis of Algorithms—Discrete.
Funding: This study is based on work supported by Laboratory Directed Research and Development funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy [Contract DE-AC02-06CH11357] and work partially supported by the U.S. Department of Energy Advanced Grid Modeling Program [Grant DE-OE0000875].

Keywords: security-constrained unit commitment • mixed-integer linear programming • machine learning

1. Introduction

Security-constrained unit commitment (SCUC) is one of the most fundamental optimization problems in power systems and electricity markets, being solved a number of times daily by major reliability coordinators, independent system operators (ISOs), and regional transmission organizations (RTOs) to schedule 3.8 trillion kilowatt-hours (kWh) of energy production and clear a $400 billion electricity market annually in the United States (Energy Information Administration (EIA) 2018). The problem asks for the most cost-efficient schedule for power generation units under a large number of physical, operational, and economic constraints. Among other requirements, a feasible generation schedule must guarantee not only that enough power is being produced during each hour to satisfy the demand but also that no transmission lines are overloaded during the delivery of the electric power. Furthermore, the schedule must be robust against small variations in demand and against the unexpected loss of a small number of network elements, such as generation units and transmission lines.

The computational performance of SCUC is an extremely important practical issue given the very short clearing windows enforced in most electricity markets. For instance, in the Midcontinent Independent System Operator (or MISO, one of the largest electricity markets in North America), systems operators only have three or four hours after receiving all bids and offers to produce a low-cost and fair generation schedule. During this short time window, SCUC must be solved multiple times under a number of different scenarios. Using state-of-art software, realistic large-scale instances can be usually solved

to 0.1% optimality within approximately 20 minutes of running time (Chen et al. 2016). Improvements in computational performance would allow markets to implement numerous enhancements that could bring significant economic benefits and improved market efficiency, such as more accurate modeling of enhanced combined cycle units, higher resolution for production cost curves, subhourly dispatch, and longer planning horizons, among others. Alternatively, a simple reduction of the optimality gap to 0.05% without sacrificing computational running times could lead to significant cost savings, considering the $400 billion market size in the United States alone.

In the past, SCUC has been solved using various optimization techniques, including priority lists (Burns and Gibson 1975), dynamic programming (Lowery 1966), and Lagrangian relaxation (Merlin and Sandrin 1983). Nowadays, SCUC is most commonly formulated as mixed-integer linear programming (MIP). The introduction of MIP solvers allowed system operators to easily model more realistic constraints that had been traditionally hard to enforce with previous methods and have benefited from the constantly advancing capabilities of modern solvers. Since the first MIP formulations, proposed in the 1960s (Garver 1962), a number of alternative formulations and polyhedral results have been described (Lee et al. 2004, Rajan et al. 2005, Ostrowski et al. 2012, Morales-España et al. 2015, Damcı-Kurt et al. 2016, Pan and Guan 2016, Gentile et al. 2017, Atakan et al. 2018, Knueven et al. 2018). There has also been active research on more practical techniques for improving the computational performance of the MIP approach, such as fast identification of redundant constraints (Zhai et al. 2010, Ardakani and Bouffard 2015, Xavier et al. 2019) and decomposition methods (Feizollahi et al. 2015, Kim et al. 2018), among others.

One aspect of SCUC that has been overlooked in previous research is that in most practical settings, this problem is repeatedly solved with only minor changes in input data, often multiple times per day. Most of the system characteristics, such as the parameters describing the generation units or the topology of the transmission network, are kept almost entirely unchanged from solve to solve. Although modern MIP solvers have the ability to reuse previous solutions as warm starts, it is well known that for SCUC, providing the previous-day optimal solution brings only minor performance improvements (Chen et al. 2016). In this work, we propose the use of machine learning (ML) to effectively extract information from previously solved instances and show how to leverage this information to significantly accelerate the solution of similar instances in the future.

ML is a discipline that studies how algorithms can automatically learn from large amounts of data and subsequently use the acquired knowledge to make decisions quickly and accurately. In the 1990s and early 2000s, researchers experimented with artificial neural networks (ANNs) to solve simpler variants of SCUC in small-scale instances (Sasaki et al. 1992, Wang and Shahidehpour 1993, Walsh and O'Malley 1997, Liang and Kang 2000). Even when considering very simplified versions of the problem, obtaining sufficiently high-quality solutions to SCUC via ANNs proved very challenging, and the approach failed to replace existing methods. Similar explorations with evolutionary algorithms proved equally challenging (Juste et al. 1999).

In more recent years, there has been a growing interest, within the broader mathematical optimization community, in applying ML techniques to enhance the performance of current MIP solvers. ML has been used, for example, to automatically construct branching strategies (Khalil et al. 2016, Alvarez et al. 2017, Lodi and Zarpellon 2017), to better parallelize branch-and-bound methods (Alvarez et al. 2014), to decide when to run primal heuristics (Khalil et al. 2017b), to predict resolution outcome (Fischetti et al. 2019), to decide solver parameters (Bonami et al. 2018), and to automatically construct decompositions (Basso et al. 2020). More broadly, there has been a renewed interest in using ML to tackle hard combinatorial problems. To give a few examples, deep learning and reinforcement learning have been used to produce heuristic solutions for optimization problems over graphs (Khalil et al. 2017a, Deudon et al. 2018, Kool et al. 2018) and for stochastic integer programs (Nair et al. 2018). We refer to Bengio et al. (2018) for a more complete survey. From a more abstract perspective, there has been research on integrating the fields of predictive, prescriptive, and descriptive analytics (Lombardi et al. 2017, Bertsimas and Kallus 2019). More directly related to our work, statistical learning has also been applied to the direct current (DC) optimal power flow problem (Misra et al. 2018).

In this work, instead of completely replacing existing MIP methods by ML models, as done in previous research with ANNs and genetic algorithms, we propose the use of ML techniques to significantly enhance the warm-start capabilities of modern MIP solvers when solving SCUC. More specifically, we develop three ML models that can automatically identify different types of patterns in previously solved instances and subsequently use these patterns to improve MIP performance. The first model is designed to predict, based on statistical data, which constraints are necessary in the formulation and which constraints can be safely omitted. The second model constructs, based on a large number of previously

obtained optimal solutions, a (partial) solution that is likely to work well as a warm start. The third model identifies, with very high confidence, a smaller-dimensional affine subspace where the optimal solution is likely to lie, leading to the elimination of a large number of decision variables and significantly reducing the complexity of the problem.

We start, in Section 2, by presenting the MIP formulation of SCUC and a brief introduction to the ML concepts we use. In Section 3, we formalize the setting in which the learning takes place, and we describe the proposed learning strategies. In Section 4, we evaluate the practical impact of the proposed ML approach by performing extensive computational experiments on a diverse set of large-scale and realistic SCUC instances, containing up to 6,515 buses and 1,388 units, under realistic uncertainty scenarios. Using the proposed technique, we show that SCUC can be solved on average 4.3 times faster than conventional methods with optimality guarantees and 10.2 times faster without optimality guarantees but no observed reduction in solution quality. We also conduct some experiments where the training and test samples are drawn from similar but not exactly the same distribution to provide some evidence that the method is somewhat robust against moderate data-set shifts. Finally, in Section 5, we discuss some limitations of the method and directions for future work. Although the techniques are presented in the context of one particular application, they can be easily adapted to other challenging optimization problems.

2. Background

2.1. MIP Formulation of SCUC

Unit commitment (UC) refers to a broad class of optimization problems dealing with the scheduling of power-generating units (Cohen and Sherkat 1987). For each hour in the planning horizon, the problem is to decide which units should be operational and how much power they should produce (Ostrowski et al. 2012). Each unit typically has minimum and maximum power outputs, a quadratic production cost curve, and fixed startup and shutdown costs. Other constraints typically include ramping rates, which restrict the maximum variation in power production from hour to hour, and minimum-up and minimum-down constraints, which prevent units from starting and shutting down too frequently. The SCUC problem is a subclass of UC that, in addition to all the constraints previously mentioned, also guarantees the deliverability of power by enforcing that the power flow in each transmission line does not exceed its safe operational limits (Shaw 1995). To guarantee sufficient security margin in the case of component failure, the problem considers transmission constraints not only under normal operating conditions but also when there is one transmission line failure in the system (so-called $N-1$ transmission contingency; Batut and Renaud 1992). In this way, even if a transmission line unexpectedly fails and the power flow in the network changes, there will be no violations of the transmission line thermal limits, which is a reliability requirement enforced by the North American Electric Reliability Corporation (NERC 2019).

A number of MIP formulations and strong, valid inequalities for SCUC have been proposed. In this work, we use the formulation proposed in Morales-España et al. (2013) because it presents good computational performance even when transmission and $N-1$ security constraints are present. The full MIP formulation is shown in the appendix. Here we present a simplified summary containing key decision variables and constraints that are most influential to the computational performance of the problem. We note, however, that the methods we present in later sections make very few assumptions about the formulation and can still be used with additional decision variables and constraints.

Consider a power system composed of a set B of buses, a set G of generators, and a set L of transmission lines. Let T be the set of hours within the planning horizon. For each generator $g \in G$ and hour $t \in T$, let x_{gt} be a binary decision variable indicating whether the generator is operational and y_{gt} be a continuous decision variable indicating the amount of power being produced. The problem can then be formulated as

$$\text{minimize} \sum_{g\in G} c_g(x_{g\bullet}, y_{g\bullet}), \tag{1}$$

$$\text{subject to } (x_{g\bullet}, y_{g\bullet}) \in \mathcal{G}_g, \qquad \forall g \in G, \tag{2}$$

$$\sum_{g\in G} y_{gt} = \sum_{b\in B} d_{bt}, \qquad \forall b \in B, t \in T, \tag{3}$$

$$-F_l^0 \le \sum_{b\in B} \delta_{lb}^0 \left(\sum_{g\in G_b} y_{gt} - d_{bt} \right) \le F_l^0, \qquad \forall l \in L, t \in T, \tag{4}$$

$$-F_l^c \le \sum_{b\in B} \delta_{lb}^c \left(\sum_{g\in G_b} y_{gt} - d_{bt} \right) \le F_l^c, \qquad \forall c \in L, l \in L, t \in T, \tag{5}$$

$$x_{gt} \in \{0,1\}, \qquad \forall g \in G, t \in T, \tag{6}$$

$$y_{gt} \ge 0, \qquad \forall g \in G, t \in T. \tag{7}$$

In the objective function, c_g is a piecewise-linear function that includes the startup, shutdown, and production costs for generator g. The notation $(x_{g\bullet}, y_{g\bullet})$ is a shorthand for the vector $(x_{gt_1}, \ldots, x_{gt_k}, y_{gt_1}, \ldots, y_{gt_k})$, where $\{t_1, \ldots, t_k\} = T$. Constraint (2) enforces a number of constraints that are local for each generator, including production limits, minimum-up and minimum-down

running times, and ramping rates, among others. Constraint (3) enforces that the total amount of power produced equals the sum of the demands d_{bt} at each bus. Constraint (4) enforces the deliverability of power under normal system conditions. The set G_b denotes the set of generators attached to bus b. We recall that power flows are governed by physical laws (i.e., Ohm's law and Kirchhoff's law). Using the DC linearization of these laws, it is possible to express the thermal limits for each transmission line as (4), where δ^0_{bl} are real numbers known as *injection shift factors* (ISFs). Similarly, Constraints (5) enforce the deliverability of power under the scenario that line c has had an outage. The bounds and the ISFs may be different under each outage scenario.

To improve the strength of the linear relaxation, the full formulation contains additional auxiliary binary decision variables. However, once x_{gt} is determined, all the other binary variables are immediately implied. Furthermore, once all the x_{gt} variables are fixed, the problem reduces to a linear programming problem (known as *economic dispatch*), which can be solved efficiently.

Constraints (4) and (5) have a significant impact on the computational performance of SCUC. The total number of such constraints is quadratic on the number of transmission lines. Because large power systems can have more than 10,000 lines, the total number of constraints can easily exceed hundreds of millions. To make matters worse, these constraints are typically very dense, which makes adding them all to the formulation impractical for large-scale instances not only because of degraded MIP performance but also because of memory requirements. Fortunately, it has been observed that enforcing only a very small subset of these constraints is already sufficient to guarantee that all the remaining constraints are automatically satisfied (Bouffard et al. 2005). Identifying this critical subset of constraints can be very challenging and usually involves either solving a relaxation of SCUC multiple times (Tejada-Arango et al. 2018, Xavier et al. 2019) or solving auxiliary optimization problems (Zhai et al. 2010, Ardakani and Bouffard 2015). In practice, system operators still rely on experience and external processes to preselect a subset of constraints to include in the optimization problem.

2.2. ML Classifiers

In this work, we use two classical supervised learning methods for binary classification: *k-nearest neighbors* (kNN) and *support vector machines* (SVMs). Given a training data set containing labeled examples $(p^1, h_1), \ldots, (p^s, h_s) \in \mathbb{R}^d \times \{-1, 1\}$, each method constructs a function $\phi : \mathbb{R}^d \to \{-1, 1\}$ that can be used to label new samples $\tilde{p} \in \mathbb{R}^d$. In the following, we present a brief description of each method. For a more complete description, we refer to Alpaydin (2014).

kNN works by identifying the most similar examples in the training data set and conducting a simple majority vote. More precisely, given a point $\tilde{p} \in \mathbb{R}^d$, the method first sorts the labeled examples by increasing distance $\|p^i - \tilde{p}\|$, according to some norm, to obtain a sequence $(p^{i_1}, h_{i_1}), \ldots, (p^{i_s}, h_{i_s})$. Then it assigns label "1" to $\tilde{p}$ if $\frac{h_{i_1} + \ldots h_{i_k}}{k} > 0$ and label "−1" otherwise. More efficient implementations, which avoid sorting the training data set for each evaluation, have also been described. In Sections 3.1 and 3.2, we use variations of this method for prediction of redundant constraints and warm starts.

SVMs construct a hyperplane that best separates training examples into two convex regions, according to their labels, and use this hyperplane for further classification. More precisely, the method first finds a hyperplane $(\pi, \pi_0) \in \mathbb{R}^d \times \mathbb{R}$ by solving the quadratic optimization problem

$$\begin{aligned} \underset{\pi, \pi_0, \alpha}{\text{minimize}} \quad & \frac{1}{2}\pi^T\pi + \frac{C}{s}\sum_{i=1}^{s} \alpha_i \\ \text{subject to} \quad & h_i\left(\pi^T p^i - \pi_0\right) \geq 1 - \alpha_i, && i = 1, \ldots, s, \\ & \alpha_i \geq 0, && i = 1, \ldots, s, \end{aligned}$$

where C is a configurable penalty term. Then it assigns the label "1" to $\tilde{p}$ if $\pi^T\tilde{p} \geq \pi_0$ and "−1" otherwise. This description is commonly referred to *SVMs with linear kernels*. Nonlinear extensions have also been described. In Section 3.3, we use SVMs to predict affine subspaces.

2.3. Model Evaluation and Cross-Validation

To evaluate the performance of a prediction method on a specific training data set, we use *k-fold cross cross-validation*. Given a training data set $\mathcal{D} = \{(p^1, h_1), \ldots, (p^s, h_s)\}$, first we partition $\mathcal{D}$ into k disjoint subsets $\mathcal{D}_1, \ldots, \mathcal{D}_k$ of equal size (we assume that s is divisible by k) and train k predictors $\phi_1, \ldots, \phi_k$ using the set $\bigcup_{j \in \{1,\ldots,k\} \setminus \{i\}} \mathcal{D}_j$ as a training data set for predictor ϕ_i. The performance of the method on $\mathcal{D}$ is then defined as

$$\sum_{i=1}^{k} \sum_{(p,h) \in \mathcal{D}_i} \frac{f\left(\phi_i(p), h\right)}{s},$$

where $f : \{-1, 1\}^d \times \{-1, 1\}^d \to [0, 1]$ is a performance measure. The idea, for each sample $(p, h) \in \mathcal{D}_i$, is to compare the actual label h against the predicted label $\phi_i(p)$ produced by a predictor ϕ_i that did not have access to the pair (p, h) during training. In this work,

we use two classical performance measures: *precision* and *recall*, given, respectively, by

$$f_{\text{precision}}\left(\tilde{h}, h\right) = \frac{\left|\left\{j : \tilde{h}_j = 1 \wedge h_j = 1\right\}\right|}{\left|\left\{j : \tilde{h}_j = 1\right\}\right|},$$

$$f_{\text{recall}}\left(\tilde{h}, h\right) = \frac{\left|\left\{j : \tilde{h}_j = 1 \wedge h_j = 1\right\}\right|}{\left|\left\{j : h_j = 1\right\}\right|}.$$

3. Setting and Learning Strategies

In this section, we formalize the setting in which the learning takes place, in addition to our proposed learning strategies. The theoretical setting that follows was designed to match the realistic conditions that energy market operators face on a daily basis.

Assume that every instance $I(p)$ of SCUC is completely described by a vector of real-valued parameters p. These parameters include only the data that are unknown to the market operators the day before the market-clearing process takes places. For example, they may include the peak system demand and the production costs for each generator. In this work, we assume that network topology and the total number of generators are known ahead of time, so there is no need to encode them via parameters. We may therefore assume that $p \in \mathbb{R}^n$, where n is fixed. We are given a set of training instances $I(p^1), \ldots, I(p^s)$ that are similar to the instance we expect to solve during actual market clearing. These training instances either can come directly from historical data, if enough similar instances have been solved in the past or can be artificially generated by sampling the probability distribution of p. Knowledge about this distribution is not an unreasonable assumption because market operators have been solving SCUC daily for years and have large amounts of accumulated data that can be readily analyzed.

We also assume that we have at our disposal a customized MIP solver that can receive, in addition to the instance $I(p)$ to be solved, a vector of hints. These hints may affect the performance of the MIP solver and hopefully will improve its running time. The hints may also affect the quality or optimality of the solutions produced by the MIP solver. Examples of hints may include a list of warm starts, a list of variables that should be fixed, or a list of initial constraints to enforce.

The day before market clearing, a *training phase* takes place. During this phase, our task is to construct a prediction function ϕ that maps each possible parameter $p \in \mathbb{R}^n$ into a vector of hints. In order to build this function, we solve the training instances $I(p^1), \ldots, I(p^s)$, capturing any data we may consider useful. During actual market clearing, when we are under time pressure to solve new instances of the problem, the *test phase* begins. In this phase, one particular vector of parameters $\tilde{p}$ is given, the instance $I(\tilde{p})$ is constructed, and our goal is to solve $I(\tilde{p})$ very quickly. To do this, the vector of hints $\phi(\tilde{p})$ is computed, and the pair $(I(\tilde{p}), \phi(\tilde{p}))$ is given to the MIP solver. The total running time is measured, including the time taken to construct the vector of hints from the parameters and the time taken by the MIP solver to solve the instance.

In the following, we propose three different ML models that target different challenges of solving SCUC. The first model is focused on predicting violated transmission and $N-1$ security constraints. The second model focuses on producing initial feasible solutions, which can be used as warm starts. The third model aims at predicting an affine subspace where the solution is likely to lie.

3.1. Learning Violated Transmission Constraints

As explained in Section 2.1, one of the most complicating factors when solving SCUC is handling the large number of transmission and security constraints that need to be enforced. Our first model is designed to predict which transmission constraints should be initially added to the relaxation and which constraints can be safely omitted.

As baseline, we use the iterative contingency screening algorithm presented in Xavier et al. (2019), which was independently implemented and evaluated at MISO, where it presented better performance than internal methods. The method works by initially solving a relaxed version of SCUC where no transmission or $N-1$ security constraints are enforced. At each iteration, all violated constraints are identified, but only a small and carefully selected subset is added to the relaxation. The method stops when no further violations are detected, in which case the original problem is solved.

In the following, we modify this baseline method to work with hints from an ML predictor. Let L be the set of transmission lines. We recall that each transmission constraint takes the form

$$-F_l^c \le \sum_{b \in B} \delta_{lb}^c \left(\sum_{g \in G_b} y_{gt} - d_{bt} \right) \le F_l^c, \quad \forall l \in L, \; c \in L \cup \{0\}, t \in T.$$

We assume that the customized MIP solver accepts a vector of hints

$$h_{l,c,t} \in \{\texttt{ENFORCE}, \texttt{RELAX}\}, \quad \forall l \in L, c \in L \cup \{0\}, t \in T,$$

indicating whether the thermal limits of transmission line l, under contingency scenario c, at time t, are likely to be binding in the optimal solution. Given these hints, the solver proceeds as described in Algorithm 1.

In the first iteration, only the transmission constraints specified by the hints are enforced. For the remaining iterations, the solver follows exactly the same strategy as the baseline method and keeps adding small subsets of violated constraints until no further violations are found. If the predictor is successful and correctly predicts a superset of the binding transmission constraints, then only one iteration is necessary for the algorithm to converge. If the predictor fails to identify some necessary constraints, the algorithm may require additional iterations but still returns the correct solution. The predictor should still strive to keep a low false-positive rate because the inclusion of too many redundant constraints during the first iteration of the procedure may significantly slow down the MIP optimization time.

Algorithm 1 (Security-Constrained Unit Commitment)

1: Create a relaxation of SCUC without Constraint (4) or (5).
2: Query transmission predictor and receive vector of hints h.
3: **for** $(l,c,t) \in L \times (L \cup \{0\}) \times T$, **do**
4: If $h_{l,c,t} =$ `ENFORCE`, add corresponding Constraint (4) or (5).
5: Solve the current relaxation.
6: **for** $(l,c,t) \in L \times (L \cup \{0\}) \times T$, **do**
7: Let $f^c_{lt} = \sum_{b \in B} \delta^c_{lb}(\sum_{g \in G_b} y_{gt} - d_{bt})$.
8: Let $\gamma^c_{lt} = \max\{-f^c_{lt} - F^c_l, 0, f^c_{lt} - F^c_l\}$.
9: Let $\Gamma = \{(l,c,t) \in L \times (L \cup \{0\}) \times T : \gamma^c_{lt} > 0\}$ be the set of violated constraints.
10: **if** Γ is empty, **then return** current solution.
11: **else**
12: For every $l \in L, t \in T$, keep in Γ only the pair (l,c,t) with the highest γ^c_{lt}.
13: For every $t \in T$, keep in Γ only the 15 pairs (l,c,t) having the highest γ^c_{lt}.
14: For every violation in Γ, add the corresponding constraint to the relaxation.
15: **goto** step 5.

For this prediction task, we propose a simple learning strategy based on kNN. Let $p^1, \ldots, p^s$ be the training parameters. During training, the instances $I(p^1), \ldots, I(p^s)$ are solved using Algorithm 1, where the initial hints are set to `RELAX` for all constraints. During the solution of each training instance $I(p^i)$, we record the set of constraints $\mathcal{L}^i \subseteq L \times (L \cup \{0\}) \times T$ that was added to the relaxation by Algorithm 1. During the test, given a vector of parameters $\tilde{p}$, the predictor finds a list of k vectors $p^{i_1}, \ldots, p^{i_k}$ that are the closest to $\tilde{p}$ and sets $h_{l,c,t} =$ `ENFORCE` if (l,c,t) appears in at least p% of the sets $\mathcal{L}^{i_1}, \ldots, \mathcal{L}^{i_k}$, where p is a hyperparameter. When $p = 50$, this strategy corresponds to the traditional kNN algorithm with simple majority voting. The cost of incorrectly identifying a constraint as necessary (false positive) is typically much smaller than the cost of incorrectly identifying a constraint as redundant (false negative): in the former case, the dimension of the problem simply increases a little, whereas in the latter, an additional iteration may be necessary. For this reason, in our experiments, we try to avoid false negatives (at the cost of more false positives) by setting $p = 10$. That is, a constraint is added if it is necessary for at least 10% of the nearest training examples.

This kNN description generalizes a number of other very simple strategies. For example, when $k = 1$, the predictor looks for the most similar instance in the training data. When $k = s$, the strategy is equivalent to counting how many times each transmission constraint was necessary during the solution of the entire training data set. When $k = 1$ and $p = 0$, the predictor simply memorizes all constraints that were ever necessary during training. In Section 4.3, we run experiments for several of these variations. As we discuss later, these simple kNN strategies already performed very well in our experiments, which is the reason we do not recommend more elaborated methods.

We note that this kNN transmission predictor is also very suitable for online learning and can be used even when only a very small number of training samples is available. During the solution of the training instances, for example, after a small number of samples have been solved, subsequent samples can already use hints from this predictor to accelerate the solution process.

3.2. Learning Warm Starts

Another considerably challenging aspect of solving large-scale SCUC instances is obtaining high-quality initial feasible solutions to the problem. Over recent decades, researchers have proposed a number of strong MIP formulations for SCUC, leading to very strong dual bounds being obtained fairly quickly during the optimization process. For challenging instances, however, we have observed that a significant portion of the running time is usually spent in finding a feasible solution that has an objective value close to this dual bound. These solutions are currently typically found through generic primal heuristics included in the MIP solver, such as feasibility pump (Fischetti et al. 2005) or large neighborhood search (Shaw 1998). Modern MIP solvers also allow users to specify user-constructed solutions, which can be used by the solver to accelerate various parts of the branch-and-cut process. Very important, the solutions provided by the user do not need to be complete or even feasible. Modern solvers can solve restricted subproblems to find values of missing variables or even repair infeasible solutions using various heuristics. The strategy we present in this subsection takes advantage of these capabilities instead of trying to

replace them. Our aim is to produce, based on a large number of previous optimal solution, a (partial) solution that works well as a warm start.

For this prediction task, we propose, once again, a strategy based on kNN. Let $p^1, \ldots, p^s$ be the training parameters, and let $(x^1, y^1), \ldots, (x^s, y^s)$ be their optimal solutions. In principle, the entire list of solutions $(x^1, y^1), \ldots, (x^s, y^s)$ could be provided as warm starts to the solver; however, processing each solution requires a nonnegligible amount of time, as we show in Section 4.4. We propose instead the use of kNN to construct a single (partial and possibly infeasible) solution, which should be repaired by the solver. The idea is to set the values only for the variables that we can predict with high confidence and to let the MIP solver determine the values for all the remaining variables. To maximize the likelihood of our warm starts being useful, we focus on predicting values for the binary variables x; all continuous variables y are determined by the solver. Given a test parameter $\tilde{p}$, first we find the k solutions $(x^{i_1}, y^{i_1}), \ldots, (x^{i_k}, y^{i_k})$ corresponding to the k nearest training instances $I(p^{i_1}), \ldots, I(p^{i_k})$. Then, for each variable x_{gt}, we verify the average $\tilde{x}_{gt}$ of the values $x^{i_1}_{gt}, \ldots, x^{i_k}_{gt}$. If $\tilde{x}_{gt} > p$, where $p \in [0.5, 1.0]$ is a hyperparameter, then the predictor sets $x_{g,t}$ to one in the warm start solution. If $\tilde{x}_{gt} \leq (1 - p)$, then the value is set to zero. In any other case, the value is left undefined.

When $p = 0.5$, this strategy corresponds to the traditional kNN method, where all variables are defined, even if the neighboring solutions significantly disagree. As p increases, a higher threshold for a consensus is required, and variables for which consensus is not reached are left undefined. More complete warm starts can be processed significantly faster by the solver but have a higher likelihood of being suboptimal or irreparably infeasible. In Section 4.4, we evaluate several choices of p to quantify this trade-off. In our computational experiments, $p = 0.9$ provided the best results.

Like the transmission predictor presented previously, this warm-start predictor is also very suitable for online learning and can be used with a very small number of training samples.

3.3. Learning Affine Subspaces

Through their experience, market operators have naturally learned a number of patterns in the optimal solutions of their own power systems. For example, they know that certain power generating units (known as *base units*) will almost certainly be committed throughout the day, whereas other units (known as *peak units*) typically come online only during peak demand. These and many other intuitive patterns, however, are completely lost when SCUC is translated into a mathematical problem. Given the fixed characteristics of a particular power system and a realistic parameter distribution, new constraints probably could be safely added to the MIP formulation of SCUC to restrict its solution space without affecting its set of optimal solutions. In this subsection, we develop a ML model for finding such constraints automatically, based on statistical data.

More precisely, let $\tilde{p}$ be a vector of parameters. Our goal is to develop a model that predicts a list of hyperplanes $(h^1, h^1_0), \ldots, (h^k, h^k_0)$ such that, with very high likelihood, the optimal solution (x, y) of $I(\tilde{p})$ satisfies $\langle h^i, x \rangle = h^i_0$, for $i = 1, \ldots, k$. In this work, we restrict ourselves to hyperplanes that can be written as

$$x_{gt} = 0, \tag{8}$$

$$x_{gt} = 1, \tag{9}$$

$$x_{gt} = x_{g,t+1}, \tag{10}$$

where $g \in G$ and $t \in T$. In other words, the predictor tries to determine whether each variable x_{gt} should be fixed to zero, to one, or to the next variable $x_{g,t+1}$. Furthermore, to prevent conflicting assignments, the predictor makes at most one of these three recommendations for each variable $x_{g,t}$.

Let $\mathcal{H}$ be the set of all hyperplanes considered for inclusion. During the training phase, our goal is to construct, for each hyperplane $(h, h_0) \in \mathcal{H}$, a prediction function $\phi_{(h,h_0)} : \mathbb{R}^n \to \{\texttt{ADD}, \texttt{SKIP}\}$ that receives a vector of parameters $\tilde{p} \in \mathbb{R}^n$ and returns the label "ADD" to indicate that the equality constraint $\langle h, x \rangle = h_0$ is very likely to be satisfied at the optimal solution and "SKIP" otherwise. Given these hints and a vector of parameters p, our customized MIP solver simply adds to the relaxation all the equality constraints $\langle h, x \rangle = h_0$ such that $\phi_{(h,h_0)}(p) = \texttt{ADD}$. These constraints are added at the very beginning of the optimization process and kept until the final solution is obtained; that is, they are never removed. Although, in principle, adding these constraints could lead to suboptimal solutions, in our computational experiments, we have observed that by using a reasonably large number of training samples and by carefully constructing high-precision predictors, as described in detail later, this strategy does not lead to any noticeable degradation in solution quality while providing significant speedups.

Next, we describe how such high-precision predictors can be constructed in practice. Let $(h, h_0) \in \mathcal{H}$. Furthermore, let $I(p^1), \ldots, I(p^s)$ be the training instances, and let $(x^1, y^1), \ldots, (x^s, y^s)$ be their respective optimal solutions. For every $i \in \{1, \ldots, s\}$, let $z^i \in \{0, 1\}$ be the label indicating whether solution (x^i, y^i) lies in

the hyperplane (h, h_0) or not. That is, $z^i = 1$ if $\langle h, x^i \rangle = h_0$ and $z^i = 0$ otherwise. We also denote by $\bar{z}$ the average value of the variables $z^1, \ldots, z^s$.

The proposed algorithm for constructing the function $\phi_{(h,h_0)}$ is as follows. First, if $\bar{z}$ is very close to one, then the predictor always suggests adding this hyperplane to the formulation. More precisely, if $\bar{z} \geq z^{\text{FIX}}$, where z^{FIX} is a fixed hyperparameter, then the prediction function $\phi_{(h,h_0)}$ returns ADD for every input. Next, the algorithm verifies whether the labels are sufficiently balanced to reliably perform supervised training. More precisely, if $\bar{z} \notin [z^{\text{MIN}}, z^{\text{MAX}}]$, where z^{MIN} and z^{MAX} are hyperparameters, then the prediction function $\phi_{(h,h_0)}$ always returns SKIP. Finally, if $\bar{z} \in [z^{\text{MIN}}, z^{\text{MAX}}]$, then the algorithm constructs a binary classifier and evaluates its precision and recall on the training set using k-fold cross-validation. If the binary classifier proves to have sufficiently high precision and recall, the prediction function $\phi_{(h,h_0)}$ returns its predictions during the test phase. Otherwise, the algorithm discards the binary classifier, and the function $\phi_{(h,h_0)}$ returns SKIP for every input. The minimum acceptable recall is given by a fixed hyperparameter α^{R}. The minimum acceptable precision is computed by the expression

$$\max\{\bar{z}, 1 - \bar{z}\} \cdot \left(1 - \alpha^{\text{P}}\right) + \alpha^{\text{P}},$$

where α^{P} is a hyperparameter. Intuitively, the algorithm only accepts the trained binary classifier if it significantly outperforms a dummy classifier that always returns the same label for every input. When $\alpha^{\text{P}} = 0$, the trained binary classifier is acceptable as long as it has the same precision as the dummy classifier. When $\alpha^{\text{P}} = 1$, the classifier is only acceptable if it has perfect precision.

Table 1 shows the precise hyperparameters used in our experiments for each type of hyperplane. Because of the high dimensionality of the parameters, we do not train the binary classifiers directly with the original vector of parameters p. Instead, we propose using as training features only the following subset of parameters: the peak system load, the hourly system loads, the average production cost of generator g, and the average production costs of the remaining generators. We propose the use of SVMs (with linear kernels) because, in our experiments, these models performed better than logistic regression and random decision forests models while remaining computationally friendly. Neural networks were not considered given the small number of training samples available.

Table 1. Hyperparameters for Affine Subspace Predictor

Hyperplane	z^{FIX}	z^{MIN}	z^{MAX}	α^{R}	α^{P}
$x_{gt} = 0$	1.000	0.250	0.750	0.90	0.90
$x_{gt} = 1$	1.000	0.250	0.750	0.75	0.75
$x_{gt} = x_{g,t+1}$	0.975	0.025	0.975	0.50	0.75

Unlike the previous two predictors, the affine subspace predictor described in this subsection requires a substantial number of samples in order to give reliable predictions, making it less suitable for online learning.

4. Computational Experiments

In this section, we evaluate the practical impact of the ML models introduced in Section 3 by performing extensive computational experiments on a diverse set of large-scale and realistic SCUC instances. In Sections 4.1 and 4.2, we describe our implementation of the proposed methods, the computational environment, and the instances used for benchmark. In Sections 4.3–4.5, we evaluate the performance of the three predictors proposed in Section 3, in addition to several variations. In Section 4.6, we evaluate the out-of-distribution performance of these predictors to measure their robustness against moderate data-set shift.

4.1. Computational Environment and Instances

The three proposed ML models described in Section 3 were implemented in Julia 1.2 (Bezanson et al. 2017) and scikit-learn (Pedregosa et al. 2011). Package ScikitLearn.jl (St-Jean 2016) was used to access scikit-learn predictors from Julia, and package DataFrames.jl (White 2012) was used for tabular data manipulation. IBM ILOG CPLEX 12.8.0 (IBM 2017) was used as the MIP solver. The code responsible for loading the instances, constructing the MIP, querying the ML models, and translating the given hints into instructions for CPLEX was also written in Julia, using JuMP (Dunning et al. 2017) as the modeling language. Solver features not available through JuMP were accessed through CPLEX's C application programming interface. Training instances were generated and solved at the Bebop cluster at Argonne National Laboratory (Intel Xeon E5-2695v4 2.10 GHz, 18 cores, 36 threads, 128-GB Double Data Rate 4 synchronous dynamic random-access memory). During the training phase, multiple training instances were solved in parallel at a time on multiple nodes. To accurately measure the running times during the test phase, a dedicated node at the same cluster was used. CPLEX was configured to use a relative MIP gap tolerance of 0.1% during test and 0.01% during training.

A total of nine realistic instances from MATPOWER (Zimmerman et al. 2011) were selected to evaluate the effectiveness of the method. These instances correspond to realistic, large-scale European test systems. Table 2 presents their main characteristics, including the number of buses, units, and transmission lines. Some generator data necessary for SCUC were missing in these instances and were artificially generated based on publicly available data from PJM (2018) and

Table 2. Size of Selected Instances

Instance	Buses	Units	Lines
case1888rte	1,888	297	2,531
case1951rte	1,951	391	2,596
case2848rte	2,848	547	3,776
case3012wp	3,012	502	3,572
case3375wp	3,374	596	4,161
case6468rte	6,468	1,295	9,000
case6470rte	6,470	1,330	9,005
case6495rte	6,495	1,372	9,019
case6515rte	6,515	1,388	9,037

MISO (2018). The modified data set has been made available online (Xavier et al. 2020).

4.2. Training and Test Samples

During training, 300 variations for each of the nine original instances were generated and solved. We considered four sources of uncertainty: (1) uncertain production and startup costs, (2) uncertain geographic load distribution, (3) uncertain peak system load, and (4) uncertain temporal load profile. The precise randomization scheme is described. The specific parameters were chosen based on our analysis of publicly available bid and hourly demand data from PJM (2018) corresponding to the month of January 2017.

1. *Production and startup costs.* In the original instances, the production cost for each generator $g \in G$ is modeled as a convex piecewise-linear function described by the parameters c_g^0, the cost of producing the minimum amount of power, and $c_g^1, \ldots, c_g^k$, the costs of producing each additional megawatt of power within the piecewise interval k. In addition, each generator has a startup cost c_g^s. In our data analysis, we observed that the daily changes in bid prices rarely exceeded ±5%. Therefore, random numbers α_g were independently drawn from the uniform distribution in the interval $[0.95, 1.05]$ for each generator $g \in G$, and the costs of g were linearly scaled by this factor. That is, the costs for generator g were set to $\alpha_g c_g^0, \alpha_g c_g^1, \ldots, \alpha_g c_g^k, \alpha_g c_g^s$.

2. *Geographic load distribution.* In the original instances, each bus $b \in B$ is responsible for a certain percentage d_b of the total system load. To generate variations of these parameters, random numbers β_b were independently drawn from the uniform distribution in the interval $[0.90, 1.10]$. The percentages d_b were then linearly scaled by the β_b factors and later normalized. More precisely, the demand for each bus $b \in B$ was set to $\frac{\beta_b d_b}{\sum_{i \in b} \beta_i}$.

3. *Peak system load and temporal load profile.* For simplicity, assume that $T = \{1, \ldots, 24\}$. Let $D_1, \ldots, D_{24}$ denote the system-wide load during each hour of operation, and let $\gamma_t = \frac{D_{t+1}}{D_t}$ for $t = 1, \ldots, 23$ be the hourly variation in system load. In order to generate realistic distribution for these parameters, we analyzed hourly demand data from PJM (2018). More specifically, we computed the mean μ_t and variance σ_t^2 of each parameter γ_t for $t = 1, \ldots, 23$. To generate instance variations, random numbers γ_t' were then independently drawn from the Gaussian distribution $\mathcal{N}(\mu_t, \sigma_t^2)$. The γ' parameters only specify the variation in system load from hour to hour and are not sufficient to determine $D_1, \ldots, D_{24}$. Therefore, in addition to these parameters, a random number ρ corresponding to the peak system load $\max\{D_1, \ldots, D_{24}\}$ was also generated. In the original instances, the peak system load is always 60% of the total capacity. Based on our data analysis, we observed that the actual peak load rarely deviates more than ±7.5% from the day-ahead forecast. Therefore, to generate instance variations, ρ was sampled from the uniform distribution in the interval $[0.6 \times 0.925C, 0.6 \times 1.075C]$, where C is the total capacity. The ρ and γ' parameters are now sufficient to construct $D_1, \ldots, D_{24}$. Figure 1 shows a sample of some artificially generated load profiles.

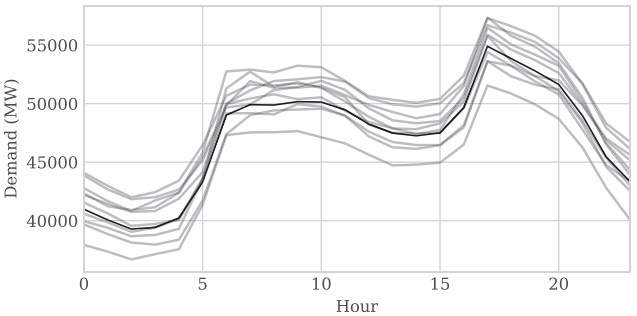
In this work, we do not consider changes to the network topology because this information is assumed to be known by the operators the day before the market clearing takes places. If the network topology is uncertain, an additional parameter $p_l \in \{0,1\}$ can be added for each transmission line $l \in L$ indicating whether the line is available or not.

During the test phase, 50 additional variations of each original instance were generated. In Sections 4.3–4.5, the test samples were generated using the same randomization scheme outline previously but with a different random seed. In Section 4.6, as described in more detailed in that section, the test samples were generated based on a similar but different distribution.

4.3. Evaluation of Transmission Predictor

We start by evaluating the performance of different strategies for predicting necessary transmission and $N-1$ security constraints. For this prediction task, we compared seven different strategies. Strategy zero corresponds to the case where no ML is used. In strategy tr:nearest, we add to the initial relaxation all

Figure 1. Sample of Artificially Generated Load Profiles



constraints that were necessary during the solution of the most similar training variation. In strategy tr:all, we add all constraints that were ever necessary during training. Strategies tr:knn:k correspond to the method outlined in Section 3.1, where $k \in \{10, 50, 300\}$ is the hyperparameter describing the number of neighbors. We recall that a constraint is initially added to the relaxation if it was necessary for at least 10% of the nearest neighbors. Because there are only 300 variations for each instance, strategy tr:knn:300 is equivalent to simply counting, over all variations, how many times each constraint was necessary. Finally, strategy tr:perf shows the performance of the theoretically perfect predictor, which can predict the exact set of constraints found by the iterative method, with perfect accuracy, under zero running time. To implement this predictor, we gave it access to the optimal solution produced by the zero strategy. For each strategy, Table 3 shows the average wall-clock running time (in seconds), the average number of constraints added to the relaxation per time period, and the average number of iterations required by Algorithm 1 to converge. Table 4 shows a more detailed breakdown of the average running times per instance.

Tables 3 and 4 show us that the prediction of necessary transmission and $N-1$ security constraints is a relatively easy prediction task, which, nevertheless, has very high impact on optimization time. Previous research has already shown that the set of necessary transmission constraints remains relatively unchanged even under significant load variations (Roald and Molzahn 2019). Our experiments here support these findings and suggest that this is true also for $N-1$ security constraints, even under significant variations in temporal load profiles and production costs. All strategies, even the most simple ones, were quite effective and significantly improved running times, with speedups ranging from 2.0 to 2.8 times. Strategy tr:nearest added the smallest number of constraints to the relaxation; however, it often failed to identify some necessary constraints, requiring additional iterations. Strategy tr:all was the most effective at reducing the number of iterations; however, it was somewhat overconservative and added significantly more constraints to the relaxation than the alternatives. The tr:knn strategies achieved a better balance between low number of constraints and low number of iterations. Among all practical strategies studied, tr:knn:300 presented the best average running times by a small margin. Overall, all methods performed very close to tr:perf. The average running time of tr:perf was slightly higher than tr:knn:300 because of the natural performance variability of solving MIPs (Lodi and Tramontani 2013), but we do not consider this difference significant.

Table 3 also shows the large negative performance impact of transmission constraints in SCUC. Comparing tr:all and tr:perf, we see that the inclusion of just 14 redundant constraints per time period (a very small number compared with the total number of constraints in the problem, which ranges from 55,000 to 230,000 after preprocessing) is already sufficient to induce noticeable slowdown in MIP performance. This is also verified by comparing the running times of instances case6468rte and case6470rte. Although these instances have roughly the same number of buses, transmission lines, and generators, the transmission system in case6468rte is significantly more congested, making it almost twice as hard to solve.

Table 3. Impact of Different Transmission Predictors on Number of Constraints Added per Time Period, Number of Iterations, and Running Time

Method	Iterations	Violations	Time (s)	Speedup
zero	4.92	11.10	226.67	1.00 times
tr:nearest	1.63	12.14	110.62	2.05 times
tr:all	1.01	25.12	91.27	2.48 times
tr:knn:10	1.06	16.36	84.53	2.68 times
tr:knn:50	1.02	15.74	82.85	2.74 times
tr:knn:300	1.02	15.92	80.36	2.82 times
tr:perf	1.00	11.10	80.94	2.80 times

Table 4. Impact of Different Transmission Predictors on Total Wall-Clock Time

	Wall-clock time (s)						
Instance	zero	tr:nearest	tr:all	tr:knn:10	tr:knn:50	tr:knn:300	tr:perf
case1888rte	35.68	14.80	10.11	9.22	9.32	9.26	9.52
case1951rte	26.07	15.69	15.34	13.00	13.05	12.68	12.51
case2848rte	39.38	25.53	18.07	17.59	17.28	17.16	18.14
case3012wp	66.11	43.95	30.75	29.22	28.71	27.96	30.21
case3375wp	124.05	84.26	82.85	69.65	62.98	64.17	65.68
case6468rte	477.48	233.31	169.87	164.18	158.82	154.04	142.21
case6470rte	290.55	122.12	98.55	97.41	89.37	89.54	97.84
case6495rte	540.38	220.54	190.93	157.92	157.37	154.92	158.64
case6515rte	440.37	235.40	205.00	202.60	208.76	193.56	193.71
Average	226.67	110.62	91.27	84.53	82.85	80.36	80.94

Table 5. Performance of Different Warm-Start Predictors

Method	Success (%)	Gap (%)	Time (s)	Speedup
tr:knn:300	0.00	—	80.36	2.82 times
ws:collect:5	92.22	0.22	64.53	3.51 times
ws:collect:15	97.41	0.18	76.37	2.97 times
ws:knn:50:50	0.00	—	80.10	2.83 times
ws:knn:50:75	46.67	0.08	72.98	3.11 times
ws:knn:50:90	100.00	0.00	52.80	4.29 times
ws:perf	100.00	0.00	46.06	4.92 times

The negative impact of transmission constraints in congested systems was also observed by Chen et al. (2016).

4.4. Evaluation of the Warm-Start Predictor

Next, we focus on the performance of the warm-start predictor described in Section 3.2. For each strategy, Table 5 shows how often the strategy was successful at producing at least one valid warm start, as well as the average wall-clock running time (in seconds) to solve the problem. The table also shows how good, in relative terms, the best warm start was compared with the optimal solution eventually found by the solver. For example, a gap of 0.25% in the table indicates that the warm start had an objective value that was 0.25% higher than the optimal objective value eventually obtained by the solver. A gap of 0% indicates that the warm start turned out to be optimal. All warm starts were provided to CPLEX with effort level CPX_MIPSTART_REPAIR, which instructs the solver to attempt to repair the warm start in case it is infeasible and to solve a sub-MIP if the values of some integer variables are missing. All other CPLEX parameters related to warm starts were left as default. Table 6 shows a more detailed breakdown of running times.

For this prediction task, we compared six different strategies. Strategy tr:knn:300, as introduced in Section 4.3, is included as a baseline and only predicts transmission constraints and not warm starts. All other strategies presented in this section are built on top of tr:knn:300; that is, prediction of transmission constraints is performed first, using tr:knn:300, and then prediction of warm starts is performed. In strategies ws:collect:n, we provide to the solver $n \in \{5, 15\}$ warm starts, containing the optimal solutions of the n nearest training variations. To maximize the likelihood of the warm starts being useful, we only provide values for the integer variables. Strategies ws:knn:$k:p$, for $k = 50$ and $p \in \{50, 75, 90\}$, correspond to the strategy described in Section 3.2. To recall, this strategy collects the solutions to the k nearest training variations and constructs a single warm start, where the value of each binary variable is provided to the solver only if the collected solutions reach a p% consensus on that variable. Strategy ws:knn:50 corresponds to the traditional kNN algorithm, using simple majority vote. Finally, strategy ws:perf shows the performance of the theoretically perfect predictor, which constructs a single warm start containing the optimal values for all binary variables under zero running time. To implement this predictor, we gave it access to the optimal solution produced by tr:knn:300.

In contrast to predicting necessary transmission constraints, predicting warm starts proved to be a significantly harder ML task. As shown in Table 6, although strategy ws:collect:5 was somewhat useful for instances with more than 6,000 buses, it was not useful for all the remaining instances. Although its achieved a speedup that was 3.5 times higher than the baseline 2.8 times, it was still well below the theoretical maximum of 4.9 times. Table 5 shows that although this strategy produced valid warm starts for 92% of the problem variations, the warm starts were of relatively low quality, with an average gap of 0.22%. The table also shows that increasing the number of added warm starts provided to the solver did little to fix this issue. The average gap of strategy ws:collect:15 was only slightly better, at 0.18%. The

Table 6. Impact of Different Warm-Start Predictors on Total Wall-Clock Time

	Wall-clock time (s)						
Instance	tr:knn:50	ws:collect:5	ws:collect:15	ws:knn:50:50	ws:knn:50:75	ws:knn:50:90	ws:perf
case1888rte	9.26	10.13	12.37	9.38	9.20	9.15	9.04
case1951rte	12.68	13.07	15.63	12.24	11.96	12.50	11.47
case2848rte	17.16	20.15	23.75	17.72	18.46	18.75	17.04
case3012wp	27.96	24.35	28.85	29.97	27.30	21.48	18.70
case3375wp	64.17	60.40	60.75	64.80	54.82	42.24	35.60
case6468rte	154.04	102.13	130.54	140.61	139.80	99.42	80.38
case6470rte	89.54	78.90	97.29	97.27	86.49	63.10	58.65
case6495rte	154.92	120.95	145.94	156.95	153.16	107.12	85.58
case6515rte	193.56	150.73	172.25	191.94	155.63	101.47	98.06
Average	80.36	64.53	76.37	80.10	72.98	52.80	46.06

time necessary to process these additional 10 warm starts, however, was significant and almost completely invalidated any potential benefits provided by them. Combining previous solutions into a complete warm start also proved challenging. Strategy ws:knn:50:50, or simple *k*NN with majority voting, proved ineffective for instances of all sizes. The main issue, in this case, was that none of the combined warm starts produced were feasible or could even be repaired by CPLEX. This difficulty in generating complete valid solutions for large-scale SCUC instances was also experienced in previous research efforts trying to solve this problem through ML methods alone.

We only obtained more significant success in warm-start prediction when we shifted some of the computational burden back to the MIP solver. Although ws:knn:50:90 does not predict values for all binary variables, we observed that the MIP solver had no difficulty in repairing these missing values. Overall, ws:knn:50:90 obtained an average speedup of 4.3 times, which is much closer to the theoretical maximum of 4.9 times than previous strategies. This strategy was very consistent and produced valid (partial) warm starts for all instance variations. More important, after being repaired by CPLEX, these warm starts turned out to be optimal in all cases.

We also experimented with other *k*NN variations, trying to produce more complete warm starts. Overall, the results were negative. We present strategy ws:knn:50:75 as an illustration. Although the (partial) warm starts provided by this strategy were of high enough quality (when they could be repaired by CPLEX), the repairing procedure failed too often, hurting the overall performance. Overall, strategy ws:knn:50:75 was not any better than ws:knn:50:90.

4.5. Evaluation of the Affine Subspace Predictor

Finally, we evaluate the performance of different affine subspace predictors. For this prediction task, we compared six different methods. Method tr:knn:300 is the *k*NN predictor for transmission constraints. This predictor does not add any hyperplanes to the relaxation and is used only as a baseline. All the following predictors were built on top of tr:knn:300. Predictor aff:svm is the affine subspace predictor described in Section 3.3, which uses SVMs and cross-validation to find hyperplanes. Predictors aff:A and aff:B follow the same strategy but use different hyperparameters. For aff:A, we set $z^{\text{FIX}} = z^{\text{MAX}} = 0.975$, $z^{\text{MIN}} = 0.025$ and keep all the other values unchanged. Our goal here is to allow prediction even under significant class imbalance; when classes are too imbalanced, we simply predict the most common class. For aff:B, we set z^{FIX}, z^{MAX} and z^{MIN} to the same values as aff:A, and in addition, we lower all precision and recall thresholds α^R and α^P to 0.50 in an attempt to make aff:A even more aggressive. Predictor aff:C uses the same hyperparameters as aff:svm, but we disable cross-validation. That is, if the classes are balanced enough, we simply train an SVM classifier and blindly use it to make predictions, without evaluating its accuracy. Finally, aff:perf represents the perfect predictor, which fixes all binary variables to their optimal values, with perfect accuracy and zero running time. To implement this predictor, we gave it access to the optimal solution produced by tr:knn:300.

Table 7 shows the average wall-clock running time (in seconds) for each predictor and the speedup relative to zero. The table also shows, in column "Feasible," how frequently the problem variations became infeasible after adding the hyperplanes. The three "Gap" columns show the distribution of relative optimality gaps. For example, a value of 0.07 in the "95%" column means that for 95% of the variations, the method produced solutions with gap 0.07% or better. The "100%" column shows the worst optimality gap observed for any variation.

Similar to the prediction of warm starts, we observe that the prediction of valid hyperplanes is a significantly hard ML task. Blindly applying SVM to the problem, without filtering for low-quality predictors, had disastrous consequences. On average, aff:C made 76% of the test variations infeasible. Even for variations that remained feasible, the additional hyperplanes caused a significant deterioration in solution quality. Predictors aff:svm, aff:A, and aff:B, which use cross-validation to filter out low-quality SVM predictors, had significantly better performance. Predictor aff:svm, our recommended predictor and the most conservative, presented an average speedup of 10.2 times and never produced invalid or suboptimal solutions during our tests. Predictor aff:A also performed very well in the typical case, with an average speedup of 11.4 times; however, it produced suboptimal solutions for a small number of instance variations (<5%), with the worst optimality gap reaching 0.17%. Predictor aff:B was overly aggressive, and although not as often as aff:C, it still produced a large

Table 7. Performance Comparison of Different Affine Subspace Predictors

		Gap (%)				
Method	Feasible (%)	80%	95%	100%	Time (s)	Speedup
tr:knn:300	100.00	0.04	0.07	0.09	80.36	2.82
aff:svm	100.00	0.03	0.05	0.08	22.24	10.19
aff:A	100.00	0.04	0.05	0.17	19.98	11.35
aff:B	72.59	0.05	0.11	0.25	12.04	18.83
aff:C	24.07	0.15	0.32	0.63	3.34	67.94
aff:perf	100.00	0.05	0.08	0.09	3.71	61.10

Table 8. Total Number of Commitment Variables: Percentage of Fixed Variables by Affine Predictor and Percentage of Correctly Identified Hyperplanes

	Commitment variables					
Instance	*Total*	*Fix Zero*	*Fix One*	*Fix Next*	*Free*	*Precision*
case1888rte	7,128.0	17.9%	52.0%	24.7%	5.4%	99.8%
case1951rte	9,384.0	20.0%	45.9%	27.8%	6.3%	99.8%
case2848rte	13,128.0	23.9%	44.8%	23.2%	8.2%	99.7%
case3012wp	12,048.0	18.5%	54.5%	21.3%	5.6%	99.8%
case3375wp	14,304.0	14.4%	57.8%	22.4%	5.5%	99.8%
case6468rte	31,080.0	7.6%	72.4%	12.8%	7.2%	99.9%
case6470rte	31,920.0	7.3%	70.3%	16.2%	6.2%	99.8%
case6495rte	32,928.0	5.8%	75.4%	13.1%	5.7%	99.9%
case6515rte	33,312.0	6.1%	75.5%	12.0%	6.4%	99.9%
Average	20,581.3	13.5%	60.9%	6.3%	19.3%	99.8%

number of infeasible problems and suboptimal solutions. Finally, we see that despite the impressive performance of aff:svm and aff:A, these predictors are still very far from the optimal speedup of 61 times obtained by aff:perf. We were unfortunately not able to reduce this gap any further.

In the following, we present a closer look at the performance of aff:svm. For each instance, Table 8 shows the total number of commitment variables in the formulation; the percentage of commitment variables x_{gt} fixed to zero, fixed to one, and fixed to the next time period (i.e., $x_{gt} = x_{g,t+1}$) by aff:svm; and the percentage of commitment variables left free. As explained in Section 3.3, each variable is fixed to at most one value, and therefore, the percentages sum to 100%.

We observe that aff:svm was able to fix a large number of commitment variables for instances of all sizes. On average, the predictor was able to eliminate 94% of the commitment variables, leaving only 6% free, a considerable reduction in problem size. To evaluate the accuracy of such variable fixing decisions, we compared the recommendations provided by the predictor against an optimal solution obtained by the MIP solver (without ML). We note that such a comparison is not entirely fair to the predictor because a large number of optimal solutions exist within the 0.1% optimality gap threshold. These solutions have slight variations and disagree with each other in a number of commitment variables. Nevertheless, on average, 99.8% of the hints provided by the predictor agreed with the optimal solution obtained by CPLEX.

Table 9 shows a summary of the results obtained. Four different strategies are presented: no ML (0), transmission prediction (tr:knn:300), transmission plus warm-start prediction (ws:knn:50:90), and transmission plus affine subspace prediction (aff:svm). Figure 2 shows the same results in graphical form, along with error bars representing 95% confidence intervals. Overall, using different ML strategies, we were able to obtain a 4.3 times speedup while maintaining optimality guarantees and a 10.2 times speedup without guarantees but with no observed reduction in solution quality.

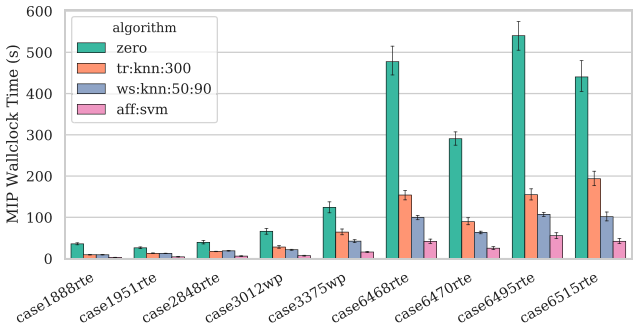
4.6. Out-of-Distribution Evaluation

In previous sections, we evaluated the performance of the ML predictors under the assumption that test and training samples are drawn from exactly the same

Table 9. Impact of ML Predictors on Running Time (In-Distribution)

	zero	tr:knn:300		ws:knn:50:90		aff:svm	
Instance	Time	Time	Speedup	Time	Speedup	Time	Speedup
case1888rte	35.68	9.26	3.85 times	9.15	3.90 times	2.83	12.61 times
case1951rte	26.07	12.68	2.06 times	12.50	2.09 times	4.19	6.22 times
case2848rte	39.38	17.16	2.29 times	18.75	2.10 times	5.96	6.61 times
case3012wp	66.11	27.96	2.36 times	21.48	3.08 times	7.03	9.41 times
case3375wp	124.05	64.17	1.93 times	42.24	2.94 times	15.88	7.81 times
case6468rte	477.48	154.04	3.10 times	99.42	4.80 times	41.76	11.43 times
case6470rte	290.55	89.54	3.24 times	63.10	4.60 times	25.14	11.56 times
case6495rte	540.38	154.92	3.49 times	107.12	5.04 times	55.52	9.73 times
case6515rte	440.37	193.56	2.28 times	101.47	4.34 times	41.88	10.52 times
Average	226.67	80.36	2.82 times	52.80	4.29 times	22.24	10.19 times

Figure 2. (Color online) Running Time



probability distribution. This is a common assumption in the literature of predictive modeling, but it may not be entirely realistic, especially when there is a limited amount of historical data or when the distribution shifts over time. In this section, we evaluate the computational performance of the predictors when the test and training distributions are somewhat similar but not exactly the same. This scenario is known in the literature as *data-set shift* (Quionero-Candela et al. 2009). Our goal here is simply to evaluate whether the performance of the predictors dramatically deteriorates under moderate data-set shift and not to establish that the predictors are completely robust against such changes. We still expect the predictors to perform the best and most reliably when the training and test distributions are as similar as possible.

In the following, we use the same training set as before, where the 300 training samples are drawn from the distribution described in Section 4.2, but we now draw the test samples from a modified distribution constructed as follows:

1. In the original distribution, production and startup cost multipliers are drawn from the uniform distribution in the interval $[0.95, 1.05]$. We replace this by the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ with mean $\mu = 1.05$ and standard deviation $\sigma = 0.017$ so that multipliers now fall, with 99.7% likelihood, in the interval $[1.00, 1.10]$ but may occasionally fall outside. That is, we assume that during training, we correctly identified the size of the interval where multipliers are likely to fall but incorrectly identified the mean and distribution, and we did not consider the possibility of outliers.

2. For geographic load distribution, we similarly replace the uniform distribution in the interval $[0.90, 1.10]$ by the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ with mean $\mu = 1$ and standard deviation $\sigma = 0.033$ so that multipliers fall, with high likelihood, in the interval $[0.90, 1.10]$. Modifying the mean has no impact in this case because load percentages are later normalized, as described in Section 4.2. Although the interval is the same, this distribution is still significantly different, and the test samples may now include outliers.

3. For peak demand, we replace the uniform distribution in the interval $[0.555, 0.645]$ with the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ with mean $\mu = 0.60 \times 1.03$ and standard deviation $\sigma = 0.009$ so that multipliers are likely to fall in the interval $[0.591, 0.645]$. That is, we assume that during test, the mean peak demand is 3% higher than during training and that the distribution is significantly different. We still assume that the multipliers fall, with very high likelihood, within an interval that was explored during training but may now contain outliers outside this interval.

These modifications can be categorized as *covariate shift* because we only modify the probability of a certain instance variation being selected. As explained in Section 3, because we focus on the scenario where the training takes places the day before market clearing and we assume that training and test samples are solved using the same mathematical model, we have no *prior probability shift*, and because new instances can still be encoded using the same set of parameters, we have no *concept shift*.

Table 10 summarizes the performance of predictors tr:knn:300, ws:knn:50:90, and aff:svm for this modified data set. The table shows the average wall-clock time (in seconds) necessary to solve each group of instances and the speedup relative to zero. Compared

Table 10. Impact of ML Predictors on Running Time (Out of Distribution)

	zero	tr:knn:300		ws:knn:50:90		aff:svm	
Instance	Time	Time	Speedup	Time	Speedup	Time	Speedup
case1888rte	44.79	8.98	4.99 times	8.70	5.15 times	2.56	17.47 times
case1951rte	29.76	14.20	2.10 times	14.12	2.11 times	4.81	6.19 times
case2848rte	43.44	18.19	2.39 times	17.87	2.43 times	6.42	6.77 times
case3012wp	75.84	32.07	2.36 times	23.78	3.19 times	8.17	9.28 times
case3375wp	145.45	77.49	1.88 times	58.74	2.48 times	19.31	7.53 times
case6468rte	426.24	169.92	2.51 times	132.32	3.22 times	49.72	8.57 times
case6470rte	268.15	91.54	2.93 times	68.80	3.90 times	27.61	9.71 times
case6495rte	512.63	179.08	2.86 times	133.24	3.85 times	72.11	7.11 times
case6515rte	406.99	189.41	2.15 times	91.19	4.46 times	36.48	11.16 times
Average	217.03	86.76	2.50 times	60.97	3.56 times	25.24	8.60 times

with Table 9, we observe that this moderate data-set shift caused some performance deterioration in all predictors, as expected, but nothing very extreme. The speedups of tr:knn:300, ws:knn:50:90, and aff:svm dropped from 2.8, 4.3, and 10.2 times, respectively, to 2.5, 3.6, and 8.6 times. Even with a degraded training set, the predictors still provided very strong performance benefits.

Regardless of the quality of the training data set, we recall that tr:knn:300 and ws:knn:50:90 maintain all MIP optimality guarantees and therefore can never negatively affect solution quality. The only potential negative impact of data-set shift, for these predictors, is a reduced speedup. For predictor aff:svm, by contrast, a sufficiently large data-set shift can potentially compromise both speed and solution quality. To evaluate the impact of data-set shift on the quality of the solutions produced by aff:svm, we present in Table 11 the distribution of their relative optimality gaps. A value of 0.08 in column "95%," for example, means that for 95% of the variations of that instance, the relative optimality gap for aff:svm was 0.08% or better. Column "100%" shows the worst optimality gap obtained for any variation of that instance.

As Table 11 shows, even under the moderate data-set shift considered, aff:svm still produced optimal solutions in the vast majority of the cases. For eight of the nine instances considered, and particularly for all instances with more than 6,000 buses, aff:svm produced optimal solutions for all variations. For instance, in case3012wp, approximately 5% of the variations were suboptimal, with the worst relative optimality gap reaching 0.13%, which is only slightly higher than the 0.1% threshold.

5. Limitations and Future Work

In this work, we proposed three ML predictors for expediting the solution of the SCUC problem and evaluated their performance on a number of realistic large-scale instances of the problem. By predicting redundant constraints and warm starts, we obtained a 4.3 times speedup over the baseline while still maintaining all optimality guarantees. By predicting a restricted affine subspace where the solution was very likely to reside, we increased this speedup to 10.2 times. Although no optimality guarantees were produced in this case, we observed that all solutions produced in our experiments were, in fact, optimal. We also performed out-of-distribution experiments, where the training and test distribution did not exactly match. Even under this challenging scenario, the predictors presented very strong performance, indicating that they are at least somewhat robust against data-set shift.

The main limitation of our approach is that in order to train the predictors, a large number of solved instances must be available, and they need to be sufficiently similar to the instances we expect to solve in the future. In our experiments, we considered the scenario where training instances are artificially generated and solved the day before market clearing takes places. Sufficient historical data must be available to characterize the distribution of parameters that are uncertain the day before market clearing so that plausible training instances can be generated. There is also some upfront computational cost when solving the training instances, although this limitation is mitigated by the fact that (1) training instances can be solved in parallel, and the number of samples required by the proposed method is relatively small, and (2) the first two predictors presented in this work can be used in online fashion and therefore can be used to significantly accelerate the generation of the training data set itself. Future work will consider ML methods that are able to handle changes to the generation fleet so that training can be performed less frequently. Alternatively, we are exploring methods to update an existing training data set more efficiently, instead of recreating it from scratch, following a significant data-set shift. Further validation on real-world data sets is also needed. Finally, we note that the techniques presented here can be adapted to other challenging combinatorial problems.

Table 11. Solution Quality (Out of Distribution)

	Gap (%)			
Instance	50%	80%	95%	100%
case1888rte	0.03	0.04	0.08	0.09
case1951rte	0.02	0.03	0.04	0.05
case2848rte	0.03	0.04	0.08	0.10
case3012wp	0.06	0.09	0.10	0.13
case3375wp	0.03	0.04	0.05	0.06
case6468rte	0.02	0.03	0.07	0.07
case6470rte	0.03	0.04	0.05	0.07
case6495rte	0.02	0.03	0.05	0.05
case6515rte	0.02	0.03	0.04	0.05

Acknowledgments

The authors thank the anonymous referees for valuable suggestions that led to significant improvements to this paper. They also gratefully acknowledge use of the Bebop cluster in the Laboratory Computing Resource Center at Argonne National Laboratory.

Appendix

Here we present the complete MIP formulation that was used during the computational experiments in Section 4. Consider a power system composed by a set B of buses, a set G of generators, and a set L of transmission lines. Furthermore, let $T = \{1, \ldots, 24\}$ be the set of hours within the planning horizon, and let G_b be the set of generators located at bus b. Let d_{bt} be the demand (in megawatt-hours) from bus b at time t. We recall that each generator $g \in G$ has a convex production cost curve, modeled as a piecewise-linear

function with a set K of segments. For each generator $g \in G$, define the following constants:

c_g^0, cost of keeping generator operational for one hour, producing at its minimum output level

c_g^k, cost to produce each additional megawatt hour of power, for each segment $k \in K$

c_g^S, cost to start the generator up

P_g^k, amount of power available (in megawatt-hours) in segment $k \in K$

RU_g, maximum allowed rise in production (in megawatt-hours) from one hour to the next

RD_g, maximum allowed drop in production (in megawatt-hours) from one hour to the next

$P_g^{\min}$, minimum amount of power (in megawatt-hours) the generator must produce if it is operational

$P_g^{\max}$, maximum amount of power (in megawatt-hours) the generator can produce

UT_g, minimum amount of time (in hours) the generator must stay operational after being switched on

DT_g, minimum amount of time (in hours) the generator must stay offline after being switched off

For each transmission line $l \in L$, let F_l^0 be its flow limit (in megawatt-hours) under normal conditions and F_l^c be its flow limit when there is an outage online $c \in L$. Similarly, let δ_{lb}^0 and δ_{lb}^c be, respectively, the injection shift factors for line l and bus b under normal conditions and under outage online c.

As mentioned in Section 2.1, the main decision variables for this problem are $x_{gt} \in \{0,1\}$, which indicates whether generator g is operational at time t, and $y_{gt} \geq 0$, which indicates how much power (in megawatt-hours) generator g produces during time t. Other auxiliary variables are $z_{gt}, w_{gt} \in \{0,1\}$, which indicate, respectively, whether generator g is started up or shut down at time t. We also define the variables y_{gt}^k, which indicate how much power produced by generator g at time t comes from segment $k \in K$. Finally, let $r_{gt} \geq 0$ be a decision variable indicating the amount of reserve (in megawatt-hours) provided by generator g at time t, and let R_t be the minimum amount of system-wide reserve required at time t. Reserves are generation capacities kept aside to compensate for small load variations. Given these variables and constants, SCUC is formulated as a cost-minimization problem:

$$\text{minimize} \sum_{g\in G}\sum_{t\in T}\left[c_g^S z_{gt} + c_g^0 x_{gt} + \sum_{k\in K} c_g^k y_{gt}^k\right] \tag{A.1}$$

$$\text{subject to} \sum_{g\in G} y_{gt} = \sum_{b\in B} d_{bt}, \qquad \forall t \in T, \tag{A.2}$$

$$\sum_{g\in G} r_{gt} \geq R_t, \qquad \forall t \in T, \tag{A.3}$$

$$\sum_{k\in K} y_{gt}^k + r_{gt} \leq \left(P_g^{\max} - P_g^{\min}\right)x_{gt} - \left(P_g^{\max} - RU_g\right)z_{gt} - \left(P_g^{\max} - RD_g\right)w_{g,t+1}, \quad \forall g \in G, t \in \{1,\dots,23\} : UT_g > 1 \tag{A.4}$$

$$\sum_{k\in K} y_{gt}^k + r_{gt} \leq \left(P_g^{\max} - P_g^{\min}\right)x_{gt} - \left(P_g^{\max} - RU_g\right)z_{gt} - \max\{RU_g - RD_g, 0\}w_{g,t+1}, \quad \forall g \in G, t \in \{1,\dots,23\} : UT_g = 1 \tag{A.5}$$

$$\sum_{k\in K} y_{gt}^k + r_{gt} \leq \left(P_g^{\max} - P_g^{\min}\right)x_{gt} - \left(P_g^{\max} - RD_g\right)w_{g,t+1} - \max\{RD_g - RU_g, 0\}z_{gt}, \quad \forall g \in G, t \in \{1,\dots,23\} : UT_g = 1 \tag{A.6}$$

$$\sum_{k\in K} y_{g,24}^k + r_{g,24} \leq \left(P_g^{\max} - P_g^{\min}\right)x_{g,24} - \left(P_g^{\max} - RU_g\right)z_{t,24}, \qquad \forall g \in G \tag{A.7}$$

$$y_{gt} \leq y_{g,t-1} + RU_g, \qquad \forall g \in G, t \in \{2,\dots,24\}, \tag{A.8}$$

$$y_{gt} \geq y_{g,t-1} - RD_g, \qquad \forall g \in G, t \in \{2,\dots,24\}, \tag{A.9}$$

$$\sum_{i=\max\{1,t-UT_g+1\}}^{t} z_{gi} \leq x_{gt}, \qquad \forall t \in T, g \in G, \tag{A.10}$$

$$\sum_{i=\max\{1,t-DT_g+1\}}^{t} z_{gi} \leq 1 - x_{\max\{1,t-DT_g\},g}, \qquad \forall t \in T, g \in G, \tag{A.11}$$

$$-F_l^0 \leq \sum_{b\in B}\delta_{lb}^0\left(\sum_{g\in G_b} y_{gt} - d_{bt}\right) \leq F_l^0, \qquad \forall l \in L, t \in T, \tag{A.12}$$

$$-F_l^c \leq \sum_{b\in B}\delta_{lb}^c\left(\sum_{g\in G_b} y_{gt} - d_{bt}\right) \leq F_l^c, \qquad \forall c \in L, l \in L, t \in T, \tag{A.13}$$

$$y_{gt}^k \leq P_g^k, \qquad \forall k \in K, g \in G, t \in T, \tag{A.14}$$

$$y_{gt} = P_g^{\min}x_{gt} + \sum_{k\in K} y_{gt}^k, \qquad \forall t \in T, \tag{A.15}$$

$$x_{gt} - x_{g,t-1} = z_{gt} - w_{gt}, \qquad \forall g \in G, t \in \{2,\dots,24\}, \tag{A.16}$$

$$x_{gt}, z_{gt}, w_{gz} \in \{0,1\}, \qquad \forall g \in G, t \in T, \tag{A.17}$$

$$r_{gt}, y_{gt}, y_{gt}^k \geq 0, \qquad \forall k \in K, g \in G, t \in T. \tag{A.18}$$

The objective function (A.1) includes startup and production costs. Although startup costs are sometimes modeled as a stepwise function of off-time, in our test, they are modeled as constants. Equation (A.2) enforces that the total power supply equals total load. Equation (A.3) enforces a sufficient amount of reserve at each time period. Equations (A.4)–(A.7) enforce the production limits. Equations (A.8) and (A.9) enforce the ramping requirements. Equations (A.10) and (A.11) guarantee that once a generator is started or shut down, it must remain online or offline for a certain amount of time. Equations (A.12) and (A.13) require that the power flow on each transmission line does not exceed its thermal limits. Equations (A.14) and (A.15) link the variables y_{gt} and y_{gt}^k. Finally, Equation (A.16) links x_{gt}, z_{gt}, and w_{gt}.

References

Alpaydin E (2014) *Introduction to Machine Learning* (MIT Press, Cambridge, MA).

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

Alvarez AM, Wehenkel L, Louveaux Q (2014) Machine learning to balance the load in parallel branch-and-bound. Technical report, Université de Liége, Liége, Belgium.

Ardakani AJ, Bouffard F (2015) Acceleration of umbrella constraint discovery in generation scheduling problems. *IEEE Trans. Power Systems* 30(4):2100–2109.

Atakan S, Lulli G, Sen S (2018) A state transition MIP formulation for the unit commitment problem. *IEEE Trans. Power Systems* 33(1): 736–748.

Basso S, Ceselli A, Tettamanzi A (2020) Random sampling and machine learning to understand good decompositions. *Ann. Oper. Res.* 284(2):501–526.

Batut J, Renaud A (1992) Daily generation scheduling optimization with transmission constraints: a new class of algorithms. *IEEE Trans. Power Systems* 7(3):982–989.

Bengio Y, Lodi A, Prouvost A (2018) Machine learning for combinatorial optimization: a methodological tour d'horizon. Preprint, submitted November 15, https://arxiv.org/abs/1811.06128.

Bertsimas D, Kallus N (2019) From predictive to prescriptive analytics. *Management Sci.* 66(3):1025–1044.

Bezanson J, Edelman A, Karpinski S, Shah VB (2017) Julia: A fresh approach to numerical computing. *SIAM Rev.* 59(1):65–98.

Bonami P, Lodi A, Zarpellon G (2018) Learning a classification of mixed-integer quadratic programming problems. van Hoeve WJ, ed. *Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (Springer International, Cham, Switzerland), 595–604.

Bouffard F, Galiana FD, Arroyo JM (2005) Umbrella contingencies in security-constrained optimal power flow. *15th Power Systems Computation Conf., Liége, Belgium.*

Burns R, Gibson C (1975) Optimization of priority lists for a unit commitment program. *Proc. IEEE Power Engrg. Soc. Summer Meeting*, Paper A-75-453-1.

Chen Y, Casto A, Wang F, Wang Q, Wang X, Wan J (2016) Improving large scale day-ahead security constrained unit commitment performance. *IEEE Trans. Power Syst.* 31(6):4732–4743.

Cohen AI, Sherkat VR (1987) Optimization-based methods for operations scheduling. *Proc. IEEE* 75(12):1574–1591.

Damcı-Kurt P, Küçükyavuz S, Rajan D, Atamtürk A (2016) A polyhedral study of production ramping. *Math. Programming* 158(1–2):175–205.

Deudon M, Cournut P, Lacoste A, Adulyasak Y, Rousseau LM (2018) Learning heuristics for the tsp by policy gradient. van Hoeve WJ, ed. *Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (Springer International, Cham, Switzerland), 170–181.

Dunning I, Huchette J, Lubin M (2017) Jump: A modeling language for mathematical optimization. *SIAM Rev.* 59(2):295–320.

Energy Information Administration (EIA) (2018) Wholesale electricity and natural gas market data. Accessed September 27, 2018, https://www.eia.gov/electricity/wholesale.

Feizollahi MJ, Costley M, Ahmed S, Grijalva S (2015) Large-scale decentralized unit commitment. *Internat. J. Electric Power Energy Systems* 73:97–106.

Fischetti M, Glover F, Lodi A (2005) The feasibility pump. *Math. Programming* 104(1):91–104.

Fischetti M, Lodi A, Zarpellon G (2019) Learning milp resolution outcomes before reaching time-limit. Rousseau LM, Stergiou K, eds. *Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (Springer International, Cham, Switzerland), 275–291.

Garver LL (1962) Power generation scheduling by integer programming-development of theory. *Trans. Amer. Inst. Electrical Engrg Part III: Power Apparatus Systems* 81(3):730–734.

Gentile C, Morales-España G, Ramos A (2017) A tight mip formulation of the unit commitment problem with start-up and shut-down constraints. *Eur. J. Comput. Optimization* 5(1–2): 177–201.

IBM (2017) IBM ILOG CPLEX optimization studio: CPLEX user's manual, version 12, release 8. Accessed September 30, 2019, https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0.

Juste K, Kita H, Tanaka E, Hasegawa J (1999) An evolutionary programming solution to the unit commitment problem. *IEEE Trans. Power Systems* 14(4):1452–1459.

Khalil E, Dai H, Zhang Y, Dilkina B, Song L (2017a) Learning combinatorial optimization algorithms over graphs. Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R, eds. *Proc. 31st Internat. Conf. Neural Inf. Process. Systems 2017* (Curran Associates, Red Hook, NY), 6348–6358.

Khalil EB, Dilkina B, Nemhauser GL, Ahmed S, Shao Y (2017b) Learning to run heuristics in tree search. *26th Internat. Joint Conf. Artificial Intelligence, Melbourne, Australia,* 659–666.

Khalil EB, Le Bodic P, Song L, Nemhauser GL, Dilkina BN (2016) Learning to branch in mixed integer programming. Schuurmans D, Wellman M, eds. *Proc. 30th AAAI Conf. Artificial Intelligence 2016* (Association for the Advancement of Artificial Intelligence, Palo Alto, CA), 724–731

Kim K, Botterud A, Qiu F (2018) Temporal decomposition for improved unit commitment in power system production cost modeling. *IEEE Trans. Power Systems* 33(5):5276–5287.

Knueven B, Ostrowski J, Watson JP (2018) On mixed integer programming formulations for the unit commitment problem. Accessed May 21, 2020, http://www.optimization-online.org/DB_HTML/2018/11/6930.html.

Kool W, van Hoof H, Welling M (2018) Attention, learn to solve routing problems! Preprint, submitted March 22, https://arxiv.org/abs/1803.08475v3.

Lee J, Leung J, Margot F (2004) Min-up/min-down polytopes. *Discrete Optim.* 1(1):77–85.

Liang RH, Kang FC (2000) Thermal generating unit commitment using an extended mean field annealing neural network. *IEEE Proc. Generation Transmission Distribution* 147(3):164–170.

Lodi A, Tramontani A (2013) Performance variability in mixed-integer programming. *Theory Driven by Influential Applications*, TutORials in Operations Research (INFORMS, Catonsville, MD), 1–12.

Lodi A, Zarpellon G (2017) On learning and branching: a survey. *TOP* 25(2):207–236.

Lombardi M, Milano M, Bartolini A (2017) Empirical decision model learning. *Artificial Intelligence* 244:343–367.

Lowery P (1966) Generating unit commitment by dynamic programming. *IEEE Trans. Power Apparatus Systems* 85(5):422–426.

Merlin A, Sandrin P (1983) A new method for unit commitment at electricite de France. *IEEE Power Engrg. Rev.* 3(5):38–39.

MISO (2018) Energy market data. Accessed December 14, 2018, https://www.misoenergy.org/markets-and-operations/market-reports.

Misra S, Roald L, Ng Y (2018) Learning for constrained optimization: Identifying optimal active constraint sets. Preprint, submitted February 26, https://arxiv:1802.09639.

Morales-España G, Gentile C, Ramos A (2015) Tight mip formulations of the power-based unit commitment problem. *OR Spectrum* 37(4):929–950.

Morales-España G, Latorre JM, Ramos A (2013) Tight and compact MILP formulation for the thermal unit commitment problem. *IEEE Trans. Power Systems* 28(4):4897–4908.

Nair V, Dvijotham D, Dunning I, Vinyals O (2018) Learning fast optimizers for contextual stochastic integer programs. Globerson A, Silva R, eds. *Proc. Thirty-Fourth Conf. Uncertainty Artificial*

Intelligence (Association for Uncertainty in Artificial Intelligence, Corvallis, OR), 591–600.

NERC (2019) NERC reliability standards. Accessed September 27, 2019, https://www.nerc.com/pa/Stand/Pages/ReliabilityStandards.aspx.

Ostrowski J, Anjos MF, Vannelli A (2012) Tight mixed integer linear programming formulations for the unit commitment problem. *IEEE Trans. Power Systems* 27(1):39–46.

Pan K, Guan Y (2016) A polyhedral study of the integrated minimum-up/-down time and ramping polytope, Preprint, submitted April 7, https://arxiv.org/abs/1604.02184.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, et al (2011) Scikit-learn: Machine learning in Python. *J. Machine Learn. Res.* 12(85):2825–2830.

PJM (2018) Energy market data. Accessed December 14, 2018, https://www.pjm.com/markets-and-operations/energy.aspx.

Quionero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND (2009) *Data Set Shift in Machine Learning* (MIT Press, Cambridge, MA).

Rajan D, Takriti S (2005) Minimum up/down polytopes of the unit commitment problem with start-up costs. IBM research report, IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY.

Roald L, Molzahn DK (2019) Implied constraint satisfaction in power system optimization: The impacts of load variations. Preprint, submitted April 3, https://arxiv.org/abs/1904.01757.

Sasaki H, Watanabe M, Kubokawa J, Yorino N, Yokoyama R (1992) A solution method of unit commitment by artificial neural networks. *IEEE Trans. Power Systems* 7(3):974–981.

Shaw JJ (1995) A direct method for security-constrained unit commitment. *IEEE Trans. Power Systems* 10(3):1329–1342.

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. Maher M, Puget JF, eds. *Principles and Practice of Constraint Programming—CP98* (Springer, Berlin), 417–431.

St-Jean C (2016) ScikitLearn.jl. Accessed September 30, 2019, https://github.com/cstjean/ScikitLearn.jl.

Tejada-Arango DA, Sánchez-Martın P, Ramos A (2018) Security constrained unit commitment using line outage distribution factors. *IEEE Trans. Power Systems* 33(1):329–337.

Walsh M, O'Malley M (1997) Augmented hopfield network for unit commitment and economic dispatch. *IEEE Trans. Power Systems* 12(4):1765–1774.

Wang C, Shahidehpour S (1993) Effects of ramp-rate limits on unit commitment and economic dispatch. *IEEE Trans. Power Systems* 8(3):1341–1350.

White JM (2012) DataFrames.jl. Accessed September 30, 2019, https://github.com/JuliaData/DataFrames.jl.

Xavier AS, Qiu F, Shabbir A (2020) Data set: Learning to solve large-scale SCUC [Data set]. Accessed May 21, 2020, http://doi.org/10.5281/zenodo.3648686.

Xavier AS, Qiu F, Wang F, Thimmapuram PR (2019) Transmission constraint filtering in large-scale security-constrained unit commitment. *IEEE Trans. Power Systems* 34(3):2457–2460.

Zhai Q, Guan X, Cheng J, Wu H (2010) Fast identification of inactive security constraints in SCUC problems. *IEEE Trans. Power Systems* 25(4):1946–1954.

Zimmerman RD, Murillo-Sánchez CE, Thomas RJ (2011) MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Trans. Power Systems* 26(1):12–19.