

Use of Machine Learning Models to Warmstart Column Generation for Unit Commitment

Nagisa Sugishita*, Andreas Grothey*, Ken McKinnon*

2023-12-18

Abstract

The unit commitment problem is an important optimization problem in the energy industry used to compute the most economical operating schedules of power plants. Typically, this problem has to be solved repeatedly with different data but with the same problem structure. Machine learning techniques have been applied in this context to find primal feasible solutions. On the other hand, Dantzig-Wolfe decomposition with a column generation procedure has been shown to be successful in solving the unit commitment problem to tight tolerance. We propose the use of machine learning models not to find primal feasible solutions directly but to generate initial dual values for the column generation procedure. Our numerical experiments compare machine learning based methods for warmstarting the column generation procedure with three baselines: column pre-population, the linear programming relaxation and coldstart. The experiments reveal that the machine learning approaches are able to find both tight lower bounds and accurate primal feasible solutions in a shorter time compared to the baselines. Furthermore, these approaches scale well to handle large instances.

1 Introduction

The unit commitment (UC) problem is an important optimization problem in the energy industry. Its aim is to compute the optimal operating schedules of power plants for given demand over a fixed time period. This problem is solved by electricity generating companies on a daily basis to determine which generators are to be used. The timings of switching the generators on and off and the amount of power dispatched have to be optimized simultaneously. The decisions in successive time periods are coupled by ramping limits (the maximum rate of change in power output) and minimum up/downtime (the minimum number of time periods for a generator needs to stay on/off after startup/shutdown to prevent damage) constraints, and this gives rise to large-scale combinatorial problems. Due to their practical importance, they have been extensively studied over the last few decades. For a recent survey, see van Ackooij et al. (2018).

This work focuses on UC problems that are to be solved repeatedly with different data but with the same problem structure. This reflects practice: when a UC problem is solved as a day-ahead planning problem, the characterisations of generators such as generation costs and ramping rates remain the same across the days, but the problems are solved with different demand forecasts each day. This makes the problem a good candidate for the use of machine learning techniques to accelerate the solution.

*School of Mathematics, University of Edinburgh, James Clerk Maxwell Building, Edinburgh (UK), EH9 3FD, Email: n.sugishita@sms.ed.ac.uk, a.grothey@ed.ac.uk, k.mckinnon@ed.ac.uk

1.1 Literature Review

ML for Optimization: In recent years, the use of machine learning techniques in optimization has been studied extensively, in particular for mixed-integer linear programming (MILP) problems. For a survey, see Bengio et al. (2021). Hutter et al. (2011) study automatic configuration of an MILP solver using machine learning. They use a local search method to find a configuration with which the MILP solver performs well on a given set of problems. Another popular application is the acceleration of branch and bound. In branch and bound, the choice of branching variables has a significant impact on the overall solution time. Khalil et al. (2016) and Gasse et al. (2019) train machine learning models to predict the output of Strong Branching and use the trained model as a quick surrogate of Strong Branching to select the variable to branch on. Other applications of machine learning to branch and bound are summarised in a survey by Lodi and Zarpellon (2017).

An overview of applications of machine learning techniques to optimization problems specifically in the energy industry is given by Yang and Wu (2021). Various authors focus on the acceleration of the solution methods for the UC problem. Dalal et al. (2018) propose using a simple nearest neighbour method to predict the optimal cost. In the training phase, they solve training instances to optimality and create a dataset of the optimal objective values. Then, given a test instance, they retrieve the nearest training instance and use the corresponding optimal value as the prediction. Pineda and Morales (2022) extend the approach to use multiple training instances and to find a feasible solution. To solve a test instance, they retrieve a prescribed number of the nearest training instances and the corresponding optimal commitment decisions. For each of these commitments, the binary variables in the UC problem are fixed to the corresponding optimal solution and the remaining continuous variables are optimized. Then, the solution with the smallest cost among the feasible solutions is adopted. Xavier et al. (2020) use a modified nearest neighbour method to construct a partial solution. Given a test instance, the average of the solutions of the nearest training instances is computed. For each binary variable, if the corresponding average is close to 0 or 1, the variable is fixed to that value. In this way, some of the binary variables are fixed and the resulting smaller problem is passed to an MILP solver. With other enhancements, the solver finds a near-optimal solution in a short time. The focus of these studies and many of the references therein is finding good primal feasible solutions in a short time. However, these methods do not give bounds on the suboptimality of the output. Furthermore, they may require a large amount of time to build the training dataset: to get a single training sample, it is necessary to solve an optimization problem to optimality, and as the problem size becomes larger, the number of variables to be predicted increases, which may require larger data sets.

Solution Methods for UC: One popular traditional approach to UC problems is to use Dantzig-Wolfe decomposition to decompose the problem by generators (e.g., see van Ackooij et al. (2018)). The reformulated problem is then solved with a column generation procedure. This procedure can be seen as the dual of a cutting plane approach to Lagrangian relaxation, as is discussed by Briant et al. (2008). Since the UC problem is an MILP problem, the reformulation is not exact but a relaxation of the original problem. However, D. Bertsekas et al. (1983) and Bard (1988) reported that the integrality gap introduced by the reformulation (i.e., the difference between the optimal objective value and the lower bound provided by the relaxation) is typically small especially if the problem size is large. In such cases, provided that a primal heuristic finds a near-optimal feasible solution, Dantzig-Wolfe decomposition is likely to solve the UC problem

to a tight tolerance.

Recently, there has been interest in accelerating the column generation procedure using machine learning. Václavík et al. (2018) use a machine learning model to speed up the solver for the pricing subproblem. They train a regression model to predict an upper bound of the objective value of the pricing subproblem, and this upper bound is passed to the optimization solver. Shen et al. (2022) study the column generation procedure applied to a graph colouring problem. They use a machine learning model to generate a near-optimal solution to the pricing subproblem. When their method fails to find a solution to the pricing subproblem that has negative reduced cost, an optimization solver is used to solve it exactly. Morabit et al. (2021) use a machine learning model to select columns from equally-promising ones obtained by solving the pricing subproblem and add them to the restricted master problem (RMP). Their approach is especially of value when primal degeneracy is present in the problem.

Warmstarted Column Generation: In the aforementioned decomposition-based approaches, the dual values play a key role, and it is expected that using appropriate dual values as an initial point will speed up the solution method. One approach to generating such dual values is to solve an approximation of the original problem and obtain its optimal dual values. Borghetti et al. (2002) and Schulze et al. (2017) relax the integrality constraints and obtain a continuous relaxation. However, the relaxation has a similar number of variables and constraints as the original problem, and solving it even without the integrality constraints takes significant computational time. Takriti et al. (1996) drop further constraints such as minimum up/down time constraints and minimum power output constraints.

A different approach to warmstarting the column generation procedure is to pre-populate columns in the RMP as is discussed by van Hoai et al. (2005). In this approach, “useful” columns are added to the RMP first, which requires being able to generate and identify useful columns in advance. In some applications, such columns may be generated using domain-specific knowledge. Alternatively, if a family of MILP problems is being solved, the columns generated for previously solved similar instances may be added.

As we will see in our numerical experiments, when the column generation procedure is applied to the UC problem, these initialisation methods (e.g., solution of the linear programming relaxation (LPR)) can account for a significant part of the total computational time and the quality of the initialisation has a big effect on the speed of convergence of the column generation procedure. In this paper, to speed up the initialisation time, we propose to warmstart the column generation procedure using machine learning techniques. Namely, we use a machine learning model to generate initial dual values. A model is first trained to output dual values which yield a tight dual lower bound. After the training, when solving a new instance, the machine learning model is used with the problem parameter as input to generate dual values. The generated dual values are then used to warmstart the column generation procedure and the column generation procedure allows us to further tighten the lower bound and, with the aid of a primal heuristic, obtain feasible solutions. With this approach, we can exploit the strength of the machine learning techniques while maintaining the desirable property of the column generation procedure (with suitable primal heuristics), such as the capability to provide high-quality solutions with very tight, provable lower bounds. Furthermore, the dimension of the dual variables is much smaller than that of the primal variables and does not depend on the number of generators but only on the number of time periods. Therefore we may expect that learning the dual values will be easier than learning primal solutions directly.

One approach is to train a machine learning model to predict the optimal dual values from the

problem parameter values using supervised learning. In the training phase, a solution method such as the column generation procedure is run on training instances to create a dataset of optimal dual values. Then, a regression model is trained using the dataset. This is closely related to the approach studied by Pineda and Morales (2022): they train machine learning models to predict the optimal primal solution using a dataset of optimal primal solutions, whereas we train the model to predict the optimal dual values using a dataset of optimal dual values.

In an alternative approach proposed by Nair et al. (2018), a neural network is trained to maximise the dual lower bound directly, without relying on a pre-built training dataset. In their study, a dual decomposition is applied to a parametrized two-stage stochastic programming problem. Using the decomposable structure of the dual lower bound by scenarios, an efficient stochastic gradient-based method is devised to train the neural network. This only requires the solution of a single scenario subproblem in each iteration and does not require a dataset of optimal dual values.

Another possible approach is based on a surrogate model. In the approach we tested, the surrogate is a regression model trained to predict the value of the dual function given the problem parameter and dual values. When a test instance is given, the problem parameter is fixed to that of the test instance, and the dual values are varied to maximise the regression model. The dual values found in this way are used to initialise the column generation procedure. One drawback of this approach is the computational time. To obtain the dual values we need to solve an optimization problem. Although the surrogate model is cheaper to optimize than the original problem, we found it still takes a longer time than the other approaches. Given this limitation, we do not consider this approach in this paper. For detailed discussions on this topic, see the survey paper by Jones (2001).

Contributions: The main contributions of this paper are as follows. Firstly, we demonstrate the use of machine learning to predict dual values that can be used to initialise the column generation procedure. By combining the machine learning techniques and the column generation procedure, we can exploit the strength of the two; namely, fast evaluation of machine learning techniques and high accuracy of the column generation procedure with provable suboptimality. This provable suboptimality property is missing in many earlier applications of machine learning to the UC problem. Secondly, we provide comprehensive numerical experiments to compare the performance of the proposed approaches on large-scale instances. The performance is measured both in terms of the tightness of the initial lower bounds and in terms of the solution time required by the warmstarted column generation procedure to find an accurate primal solution of proven suboptimality.

The rest of the paper is structured as follows. Section 2 briefly reviews Dantzig-Wolfe decomposition and the column generation procedure. Section 3 presents methods based on machine learning models to generate initial values for the column generation procedure. In Section 4 the proposed approach is applied to large-scale UC problems. Finally, in Section 5 conclusions and further extensions of this work are presented.

2 Dantzig-Wolfe Decomposition and Column Generation

As noted in Section 1, Dantzig-Wolfe decomposition is known to be effective for the UC problem. In this section, we briefly review Dantzig-Wolfe decomposition and the column generation procedure. For further background, see Vanderbeck and Savelsbergh (2006).

Consider the following family of MILP problems parametrized by ω :

$$\begin{aligned} z(\omega) = \min_{x_1, \dots, x_G} & \sum_{g=1}^G c_g^T x_g \\ \text{s.t.} & \sum_{g=1}^G A_g x_g = a(\omega), \\ & x_g \in X_g, \quad g = 1, 2, \dots, G, \end{aligned} \quad (2.1)$$

where G is the number of subproblems, x_1, x_2, \dots, x_G are vectors of decision variables and

$$X_g = \{x_g \in \{0, 1\}^n \times \mathbb{R}^m \mid D_g x_g \leq d_g\}, \quad g = 1, 2, \dots, G.$$

Here, n and m are the number of integer and continuous variables respectively in x_g , and for each g and ω $c_g \in \mathbb{R}^{n+m}$, $A_g \in \mathbb{R}^{k \times (n+m)}$, $a(\omega) \in \mathbb{R}^k$, $D_g \in \mathbb{R}^{l \times (n+m)}$, $d_g \in \mathbb{R}^l$, where k and l denote the number in each of the corresponding constraints. We assume that X_g is non-empty and bounded for $g = 1, 2, \dots, G$, and that the problem (2.1) has a feasible solution for every ω . To reduce clutter, in what follows we drop the dependence on ω except where this might cause confusion. We also write $x = (x_1, x_2, \dots, x_G)$ and $X = X_1 \times X_2 \times \dots \times X_G$. In the UC problem, G is the number of generators, $x_g \in X_g$ corresponds to a feasible operational plan of generator g , ω is the vector of demands and the first constraint in (2.1) represents the system-wide constraints (i.e., the load balance and spinning reserve constraint). The constraint right-hand-side $a(\omega)$ has entries for the demand and spinning reserve. The complete formulation of the UC problem is given in Appendix A.

In Dantzig-Wolfe decomposition, we consider a relaxation of (2.1), referred to as the master problem (MP), in which X_g is replaced by $\text{conv}(X_g)$, the convex hull of X_g , for every g . Given the boundedness assumption on X_g the MP can be written in terms of the extreme points $\{x_{gi} \mid i \in I_g\}$ of X_g as

$$\min_p \sum_{g=1}^G \sum_{i \in I_g} c_g^T x_{gi} p_{gi} \quad (2.2)$$

$$\text{s.t.} \sum_{g=1}^G \sum_{i \in I_g} A_g x_{gi} p_{gi} = a, \quad (2.3)$$

$$\sum_{i \in I_g} p_{gi} = 1, \quad g = 1, 2, \dots, G, \quad (2.4)$$

$$p_{gi} \geq 0, \quad g = 1, 2, \dots, G, i \in I_g.$$

This is a linear programming (LP) problem with decision variables p_{gi} ($g = 1, 2, \dots, G, i \in I_g$).

Since the MP is usually too large to formulate and solve explicitly, a column generation procedure is used. The restricted master problem (RMP) is defined by replacing each I_g in the MP with a subset $\hat{I}_g \subset I_g$. We assume that the RMP is feasible (the sets \hat{I}_g must have suitable columns to satisfy the requirement of constraint (2.3)) and define y and σ_g for $g = 1, 2, \dots, G$ to be the optimal dual values to the RMP corresponding to the restricted version of constraints (2.3) and (2.4), respectively. To find columns to be added to the RMP, the following subproblems,

known as the pricing subproblems, are solved:

$$r_g(y) = \min_{x_g} \{(c_g^T - y^T A_g)x_g \mid x_g \in X_g\}, \quad g = 1, 2, \dots, G. \quad (2.5)$$

In the UC problem, this pricing subproblem is a scheduling problem of a single generator: given a reduced cost $c_g^T - y^T A_g$, it finds the cheapest operational plan for the corresponding generator. If $r_g(y) \geq \sigma_g$ for all g , the RMP already includes all relevant columns and has found the optimal solution to the MP. Otherwise, the solutions to the pricing subproblems for which $r_g(y) < \sigma_g$ are added to the RMP and the above process is repeated.

For each y , a lower bound on the optimal objective value of (2.1) can be obtained using duality. Let us denote the Lagrangian of (2.1) by

$$L(x, y) = \sum_{g=1}^G c_g^T x_g - y^T \left(\sum_{g=1}^G A_g x_g - a \right) = a^T y + \sum_{g=1}^G (c_g^T - y A_g) x_g.$$

Since the minimisation of $L(x, y)$ over $x \in X$ for any y is a relaxation of (2.1), the the optimal objective value z of (2.1) satisfies

$$z \geq \min_{x \in X} L(x, y) =: q(y), \quad (2.6)$$

for any y . From (2.5) and the definition of the Lagrangian, it follows that

$$q(y) = a^T y + \sum_{g=1}^G r_g(y). \quad (2.7)$$

We refer to this value as the *dual lower bound*.

It is known that the simple column generation approach discussed above suffers from instability. In the first few iterations, due to the poor RMP model, the column generation procedure tends to output irrelevant dual values. Vanderbeck (2005) refers to this behaviour as the “heading-in effect”. This is closely related to a well-known instability issue of the cutting-plane algorithm. See for example the discussion in Chapter XII and XV in the book by Hiriart-Urruty and Lemaréchal (1993). To tackle the issue, it is necessary to deploy some mechanism to stabilise the dual values, such as quadratic regularisation of the dual values or the box step method as described by Briant et al. (2008). In this work, quadratic regularisation of the dual values is used. We note that regularisation is also used to cope with degeneracy when the RMP is degenerate. However, in our case, we did not observe any degeneracy of the RMP when solving the UC problem. The purpose of adding the regularisation in our context is to mitigate the “heading-in effect”.

Dualizing the RMP and adding quadratic regularisation on the dual values gives

$$\begin{aligned} \max_{y, \sigma} \quad & a^T y + \sum_{g=1}^G \sigma_g - \frac{\mu}{2} \|y - \bar{y}\|_2^2 \\ \text{s.t.} \quad & \sigma_g \leq (c_g^T - y^T A_g)x_{gi} \quad g = 1, 2, \dots, G, i \in \hat{I}_g, \\ & y, \sigma : \text{free}, \end{aligned} \quad (2.8)$$

where \bar{y} is a regularisation centre and μ is a parameter used to adjust the strength of the regularisation. We refer to (2.8) as the regularised RMP. In the regularised column generation procedure, the regularised RMP is used in place of the RMP. The optimal solution to (2.8) is computed and the pricing subproblems are solved based on this solution. The regularisation centre \bar{y} is updated to the current dual values y whenever the lower bound (2.7) has improved. In the following, we normally omit the word “regularised” when we refer to the regularised RMP or the regularised column generation procedure but add the word “unregularised” when referring to the one without regularisation (i.e., $\mu = 0$).

2.1 Warmstarting the Column Generation Procedure

In the above algorithm, the dual values y are updated iteratively to provide tighter lower bounds. It can be shown that the lower bound $q(y)$ is continuous (in fact concave) in y . It follows that in the case where near-optimal dual values are used as the initial point of the algorithm, the lower bound in the first iteration will be close to the tightest. Assuming that the lower bound is sufficiently tight, the algorithm would terminate as soon as a primal feasible solution with sufficiently small suboptimality was found by primal heuristics. It is therefore expected that using near-optimal dual values for the initial dual values will help the algorithm to terminate in a shorter time.

It is worth noting that the regularisation of the dual values as discussed above is crucial in this context. Without regularisation, even if near-optimal dual values are used as the initial point, the algorithm is likely to be quite unstable, yielding dual values with poor lower bounds in the following iterations. See Briant et al. (2008) for further discussion.

As discussed in Section 1, one approach to finding good dual values is to solve an approximation of (2.1). Let us denote the linear relaxation of X_g by \bar{X}_g :

$$\bar{X}_g = \{x_g \in [0, 1]^n \times \mathbb{R}^m \mid D_g x_g \leq d_g\}, \quad g = 1, 2, \dots, G.$$

The linear programming relaxation (LPR) is obtained by replacing X_g with \bar{X}_g in (2.1). Borghetti et al. (2002) and Schulze et al. (2017) solve the LPR to optimality and use the optimal dual values to the LPR as the initial dual values for the column generation procedure. This approach does not require any training as is required in a machine learning model. On the other hand, solving the LPR takes a non-trivial amount of time.

Another approach is to initialise an unregularised RMP with “useful” columns and use the optimal dual values to the unregularised RMP as the regularisation centre of the following iteration. To this end, we need to be able to generate useful columns before running the column generation procedure. For example, if we solve similar instances sequentially, we may use the columns generated for the previous instances.

2.2 Initial Dual Lower Bounds in the Column Generation Procedure

We note that the LPR is a relaxation of (2.1) so it gives a lower bound on the optimal objective value. On the other hand, any dual values give a lower bound on the optimal objective value of (2.1) by (2.7). In particular, the optimal dual values to the LPR give such dual values. In the first iteration, the column generation procedure with the LPR initialisation evaluates this lower bound. It is of interest to compare these two lower bounds. Let \bar{x}^* and \bar{y}^* be any optimal primal

and dual values to the LPR, respectively, and let \bar{z}^* be the optimal primal and dual objective value of the LPR. Using (2.6) and the fact that $\bar{X} \supset X$, we have

$$q(\bar{y}^*) = \min_{x \in X} L(x, \bar{y}^*) \geq \min_{x \in \bar{X}} L(x, \bar{y}^*).$$

On the other hand, since \bar{y}^* is the optimal dual values to the LPR, strong duality (e.g., see D. P. Bertsekas (2009)) gives

$$\min_{x \in \bar{X}} L(x, \bar{y}^*) = L(\bar{x}^*, \bar{y}^*) = \bar{z}^*.$$

Thus, the dual lower bound $q(\bar{y}^*)$ computed using the optimal dual values to the LPR is at least as tight as (and probably tighter than) the optimal objective value of the LPR, \bar{z}^* , i.e.,

$$q(\bar{y}^*) \geq \bar{z}^*.$$

3 Machine Learning Methods to Compute Initial Dual Values

In Section 2.1, we briefly discussed existing approaches to warmstarting the column generation procedure. In this section, we consider approaches to training a machine learning model to generate initial dual values for the column generation procedure. In a practical situation, the goal is to solve the UC problem as quickly as possible. As we will observe in our numerical experiments, there is a strong connection between solution time and the tightness of the initial dual lower bound. Thus, we will train a machine learning model to output dual values that yield a tight dual lower bound (2.7), using this as a surrogate measure of solution time. Below, we consider two approaches to achieve this goal.

3.1 Machine Learning Model based on Single-Sampling Training

A simple approach to training a machine learning model is to build a dataset of optimal dual values of training instances and train a regression model to predict the optimal dual values from the problem parameter. To build a dataset, a set of training instances must be solved for example with the column generation procedure. This approach based on a dataset of optimal solutions is used to predict optimal solutions of optimal power flow problems by Guha et al. (2019), Zamzam and Baker (2020), and Owerko et al. (2020). One can use any prediction model to this end. In our numerical experiments, we will use the alternatives of a neural network model, a random forest model and a nearest neighbour model. We refer to this approach in the remainder of the paper as *single-sampling training* since this only involves the sampling of the problem parameter ω .

3.2 Machine Learning Model based on Double-Sampling

A potential drawback of the single-sampling training is the large amount of time required to solve enough problems to build a sufficiently large dataset. This is especially problematic when the problem is large. An alternative approach is to train a machine learning model to maximise the expected dual lower bound directly. This approach can exploit the decomposable structure of the dual function. This approach was introduced by Nair et al. (2018) and used to solve a parametrized two-stage stochastic programming problem.

A neural network model can be seen as a function $f(\omega, \theta) = y$ which maps the problem parameter ω and the model parameter θ to a dual value y . We aim to learn values of the model parameter θ so that given ω the model outputs a dual value y for which the dual lower bound $q(y)$ is tight. In this section, we explicitly show the dependency of the lower bound on ω as $q(\omega, y)$, which was suppressed in (2.7). Assume that the distribution of ω is given (e.g., the empirical distribution based on historical data). Our goal is to maximise the expected lower bound

$$p(\theta) = \mathbb{E}_\omega[q(\omega, f(\omega, \theta))].$$

From (2.7), it follows that

$$q(\omega, y) = a(\omega)^T y + \sum_{g=1}^G r_g(y) = \frac{1}{|G|} \sum_{g=1}^G (a(\omega)^T y + |G| r_g(y)).$$

We can interpret the final term as the expectation $\mathbb{E}_g[\tilde{q}(\omega, y, g)]$ where g is uniformly sampled from $\{1, 2, \dots, G\}$ and

$$\tilde{q}(\omega, y, g) = a(\omega)^T y + |G| r_g(y).$$

It follows that

$$p(\theta) = \mathbb{E}_{\omega, g}[\tilde{q}(\omega, f(\omega, \theta), g)].$$

$p(\theta)$ can be seen as an expectation over both ω and g . We use the stochastic gradient ascent method, the standard approach used for training a neural network. That is, we sample ω and g , compute the gradient of \tilde{q} with respect to θ and make a single gradient ascent step. We then resample ω and g and repeat the process.

The gradient of \tilde{q} with respect to θ can be computed as follows: fix problem parameter ω and subproblem index g and compute the dual values $y = f(\omega, \theta)$ and the component $\tilde{q}(\omega, y, g)$ of the dual lower bound corresponding to subproblem g . Suppose that the model output $f(\omega, \theta)$ is differentiable with respect to θ and the optimal value $r_g(y)$ of the pricing subproblem (2.5) is differentiable with respect to y (which is the case when the optimal solution to the pricing problem is unique). Then the gradient of $\tilde{q}(\omega, y, g)$ with respect to y is given by

$$\frac{\partial \tilde{q}}{\partial y} = a - |G| A_g x_g^* \tag{3.1}$$

where x_g^* is the solution to the pricing subproblem g (2.5). Using the chain rule, we obtain

$$\frac{\partial \tilde{q}}{\partial \theta} = J \frac{\partial \tilde{q}}{\partial y}, \tag{3.2}$$

where J is the Jacobian matrix of the neural network output $y = f(\omega, \theta)$ with respect to θ , which is given by automatic differentiation. In this paper, we refer to this approach as *double-sampling training* since this involves sampling both the problem parameter ω and the subproblem g .

The procedures of the single and double-sampling training are listed in Figure 1. It is important to note that each step of the double-sampling training involves evaluation of (3.2), which requires the solution of only a single pricing subproblem, which is typically substantially smaller than the original problem. On the other hand, the single-sampling training requires solving the

training UC instances to optimality. Furthermore, the single-sampling training uses the mean squared error to train a model, that is, the model is encouraged to output the dual values close to the optimal dual values. However, the dual values close to the optimal values in terms of the Euclidean distance do not necessarily lead to a tight dual lower bound. In contrast, the double-sampling training uses the dual lower bound to train the model, which is the metric we are directly interested in.

One advantage of the single-sampling training is its flexibility. The training is based on the dataset of the optimal dual values. Once the dataset is built, one can fit models with different architectures.

Single-Sampling Training

- Solve as many training instances of (2.1) with different ω as possible within the training budget.
 - Save each problem parameter ω and the corresponding near-optimal dual values.
- Train a regression model.
 - Train to predict the optimal dual values from given ω .

Double-Sampling Training

- Repeat as often as possible within the training budget:
 - Sample problem parameter ω and generator g .
 - Compute the gradient of $\tilde{q}(\omega, f(\omega, \theta), g)$ with respect to θ using (3.1) and (3.2).
 - Do a stochastic gradient step to improve $p(\theta)$.

Figure 1: Diagrams to show the training procedures. The upper one corresponds to the single-sampling training while the lower one is to the double-sampling training.

Once a model is trained with the single or double-sampling training, it can be used to compute initial dual values for the regularised column generation procedure, and it is expected that unlike solving an approximation such as the LPR the computation will be quick. Furthermore, we later observe experimentally that the trained model produces high-quality initial dual values, in terms of both the tightness of the resulting dual lower bound and the solution time of the column generation procedure when warmstarted from it.

4 Numerical Experiments

In this section, the performance of the dual initialisation methods based on machine learning models as well as the benchmark initialisation methods is evaluated on UC problems of various sizes.¹ All methods are implemented in Python. IBM ILOG CPLEX 20.1.0² is used as the optimization solver (the barrier method for the RMP and the branch and bound method for the pricing subproblems), and PyTorch and scikit-learn are used to implement the neural network models and the random forest models, respectively. The experiments are performed on a workstation with a 16-core Intel[®] Xeon[®] E5-2670 and 126 GB of RAM.

¹The source code is available at: https://github.com/nsugishita/ml_to_warmstart_cg

²<https://www.ibm.com/products/ilog-cplex-optimization-studio>

4.1 Problem

In the experiments, we consider a setup in which UC problems are solved repeatedly with a fixed set of generators but with different demand forecasts. To assess the scalability, we consider 3 different problem sizes, i.e., problems with 200, 600 and 1,000 generators. In all cases, the length of the planning horizon is 48 hours with a time resolution of 1 hour. The generator data is based on Borghetti et al. (2002). Since their sets of generators contain 200 generators at most, we combine multiple sets to create larger ones. For example, to create a UC instance with 1,000 generators, we combine 5 distinct 200-generator sets. Each generator is unique and distinct across the sets so combining these sets does not introduce symmetry. The demand data is based on the historical demand data in the UK published by National Grid ESO.³ A detailed description of the problem formulation and implementation details of the initialisation methods are given in Appendix A.

4.2 Initialisation Methods

The initialisation methods used in the experiments are described below.

4.2.1 Benchmark initialisation Methods.

The first three methods are not based on machine learning models but are evaluated as benchmarks.

Coldstart: as a naive baseline, column generation is run from initial dual values $y = 0$. This method does not require any training.

LPR: this is the method based on the LPR described in Section 2.1. Given a test instance, we first solve the LPR by CPLEX. Then, the optimal dual values are used as the initial dual values for the column generation procedure. This method does not require any training. This is the method used by Borghetti et al. (2002) and Schulze et al. (2017).

Column pre-population: this approach requires training instances to be solved beforehand. Before the evaluation, for each set of generators, as many training instances as possible are solved to 0.25% optimality in 24 hours with 8 CPU cores. The number of training instances solved is reported in Table 1. The training instances are solved using the column generation procedure with the LPR initialisation and the local search primal heuristic. Implementation details are given in Appendix B. For each training instance, the problem parameter values and all the generated columns are saved. To solve a test instance, its problem parameter values (i.e., the demands) are compared against those of the training instances. The 70 training instances with the closest problem parameter values in terms of the Euclidean distance are selected. The columns of the selected nearest training instances are retrieved and added to the RMP of the test instances. The RMP is then solved without regularisation and the optimal dual values are used as the initial dual values. In our preliminary experiments, we observed that including the pre-populated columns in the RMP increased the RMP solution times significantly and degraded the overall performance. We therefore discard these columns. The number of nearest neighbours

³<https://www.nationalgrideso.com/>

Table 1: Number of training instances solved

instance size	number of training instances solved
200	25,660
600	12,023
1000	8,588

used, i.e., 70, was chosen by exhaustive grid search on a set of validation UC instances which were different from the test instances and the training instances. That is, the performance of the column pre-population initialisation with different numbers of neighbours was evaluated and the number of neighbours with the best performance was chosen.

4.2.2 Machine Learning Methods based on Single-Sampling Training.

The next three methods are based on the single-sampling training and require a dataset of the optimal dual values of training instances. In our implementation, the datasets created for the column pre-population initialisation (where the optimal dual values are also stored) are used.

Nearest Neighbour: given a test instance, the four training instances with the closest problem parameter values are chosen, and the mean of the corresponding optimal dual values are used as initial dual values. The number of nearest neighbours (i.e., four) is chosen by grid search on validation UC instances, in the same way as the column pre-population initialisation.

Random forest: a random forest model (Briant et al. (2008)) is trained using the dataset to predict the optimal dual values given problem parameter values ω . The training of the models took less than 30 seconds in all cases. We used the default hyperparameter values of scikit-learn.

Neural Network (single-sampling): the neural network model consists of 4 hidden layers of 1000 units per layer, with skip connections between hidden layers and tanh as an activation function. The hyperparameters are chosen based on the performance on validation instances. For more details on model architecture and training procedure, see Appendix C. The time to train the neural network models was less than 15 minutes in every case.

4.2.3 Machine Learning Methods based on Double-Sampling Training.

The final method is based on the double-sampling training.

Neural Network (double-sampling): the initialisation method based on a neural network model with the double-sampling training is implemented as described in Section 3.2. The neural network model has the same architecture as the one used for the single-sampling training. For each set of generators, a single neural network model is trained with the same training time (24 hours using 8 CPU cores). See Appendix C for more detail on the training procedure.

4.3 Evaluation

The initialisation methods based on machine learning models are trained to predict dual values which give tight dual lower bounds. To evaluate the performance of the methods, we first evaluate the dual lower bound at the dual values output by the initialisation methods. Then, in the next experiment, the column generation procedure is run using the initialisation method but without any primal heuristics and the time to find a near-optimal dual lower bound is measured. In the final experiment, the column generation procedure is run with primal heuristics and the time to find a near-optimal primal feasible solution is measured.

4.3.1 Evaluation: Dual Lower Bound

To run the evaluations, 100 test instances of each size were created. For each test instance, CPLEX was used for two hours to solve it and the best lower bound found within the time limit was saved (we note that when there is an integrality gap between the Lagrangian lower bound and the optimal objective value if given sufficient time CPLEX will find a tighter lower bound than the column generation procedure). Then, the dual lower bounds computed at the dual values output by the initialisation methods were calculated by (2.7) and compared against the lower bound found by CPLEX. Table 2 shows the tightness of the dual lower bounds and the required time (all times in this section are wall clock times) to run the initialisation methods. The tightness of the dual lower bounds is reported as the average percentage gaps between the bounds. For comparison, the average gap for the objective value of the LPR (which is also a valid lower bound) is shown in the table as well (labelled as LPR objective value).

Clearly, coldstart yields poor lower bounds. The performance of LPR initialisation is significantly better than coldstart. The column pre-population initialisation method gives even better lower bounds, and in particular, is the best on 200-generator instances. However, the performance is poorer on the largest test instances. The lower bounds of the methods based on machine learning are comparable, and they give the best lower bounds on large test instances. Among these methods, the neural network with the double-sampling training performs best on the two largest instances. The computational time of the column pre-population and LPR initialisation grows significantly as the problem size increases. On the other hand, the computational time of the other methods remains small. We note that as discussed in Section 2.2 the optimal LPR objective value gives a lower bound, but we see from Table 2 that the dual lower bound evaluated at the optimal dual yields significantly tighter lower bounds.

4.3.2 Evaluation: Column Generation Procedure without Primal Heuristic

Given the dual values computed by the initialisation methods, the column generation procedure is expected to successively tighten the lower bound. Hence, if the initialisation methods give good dual values, the column generation procedure is more likely to find a near-optimal dual lower bound in a short time.

To verify this point, the column generation procedure was run with the initialisation methods, but without any primal heuristic, until dual values that yield a dual lower bound of a prescribed suboptimality (0.1%, 0.05% and 0.025%) were found or the time limit of 10 minutes was reached. We note that, without primal heuristics, the column generation procedure does not provide upper bounds, and therefore no measure of suboptimality of the lower bounds is available. We have monitored the progress of the column generation procedure (i.e., the suboptimality of

Table 2: Tightness of initial lower bound lb_1 (%) and computational time (seconds). The tightness of initial lower bounds lb_1 is measured by comparing it with the best known lower bound lb_{CPLEX} obtained by running CPLEX for 2 hours and is reported as the percentage of lb_{CPLEX} . The values reported in this table are average over 100 test instances.

method	size: 200		600		1000	
	$lb_{\text{CPLEX}} - lb_1$	time	$lb_{\text{CPLEX}} - lb_1$	time	$lb_{\text{CPLEX}} - lb_1$	time
coldstart	99.823	0.0	99.875	0.0	99.882	0.0
LPR (dual lower bound)	0.133	4.6	0.121	18.1	0.092	29.6
LPR objective value	0.193	4.6	0.195	18.1	0.155	29.6
column pre-population	0.019	15.0	0.025	24.2	0.073	34.5
nearest neighbour	0.041	<0.1	0.052	<0.1	0.059	<0.1
random forest	0.056	<0.1	0.062	<0.1	0.064	<0.1
network (single-sampling)	0.047	<0.1	0.047	<0.1	0.048	<0.1
network (double-sampling)	0.048	<0.1	0.037	<0.1	0.026	<0.1

the computed lower bounds) by comparing with the best lower bound computed by CPLEX beforehand as explained in Section 4.3.1.

Table 3 shows the average computational time and the average number of iterations to find the near-optimal dual lower bound or until the time limit of 10 minutes was reached, and the number of problems for which the column generation procedure found a near-optimal dual lower bound within the time limit. The computational time reported in this table includes the time to run the initialisation methods such as solving LPR as well as the time to solve the RMP and the pricing subproblems. For the instances that are not solved within the time limit, the time is set to be 10 minutes and the number of iterations reached by the 10-minute time limit is used.

Clearly, coldstart initialisation methods are the slowest, followed by the LPR initialisation method. The methods based on machine learning are the best. On 600-generator instances, the neural network with the double-sampling training is one of the best methods, and on 1000-generator instances, it gives the best performance on average (the standard error of the computational time was less than 4 seconds). There is a correlation between the results in Table 2 and Table 3: if an initialisation method outputs dual values with a tight lower bound, the column generation procedure tends to require fewer iterations. For example in the 200-generator and 600-generator instances, the column pre-population initialisation outputs dual values with the tightest lower bound and the column generation procedure requires a smaller number of iterations than the other methods. However, finding the initial dual values using the column pre-population approach is very slow compared to double-sampling initialisation and as a result, the solution obtained using the double-sampling initialisation is on average significantly faster.

4.3.3 Evaluation: Column Generation Procedure with Primal Heuristic

In the previous experiments, the initialisation methods were evaluated in terms of the quality of the dual lower bounds. This is a natural criterion to evaluate the methods since they are trained to maximise the dual lower bound. However, in practice, the primary interest is to solve the original UC instance by finding a good feasible solution. To see the performance of the initialisation methods in a more practical setup the column generation procedure was run with primal heuristics until a solution within 1%, 0.5% and 0.25% suboptimality was found or the time limit of 10 minutes was reached. The description of the primal heuristic used is given in

Table 3: Performance of the column generation procedure without primal heuristics. The three columns labelled as “time”, “iters” and “solved” show average computational time (seconds) and iterations, and the number of problems solved within the time limit (problems where the column generation procedure found a near-optimal lower bound), respectively.

size	method	0.1% optimality			0.05% optimality			0.025% optimality		
		time	iters	solved	time	iters	solved	time	iters	solved
200	coldstart	95.9	18.2	100	131.0	21.5	99	181.6	25.6	97
	LPR	15.1	3.2	100	23.5	5.5	100	35.8	8.4	100
	column pre-population	19.9	1.0	100	19.9	1.0	100	21.2	1.4	100
	nearest neighbour	4.3	1.0	100	8.0	2.0	100	21.6	5.3	100
	random forest	4.3	1.1	100	9.8	2.5	100	27.6	7.0	100
	network (single-sampling)	4.1	1.0	100	6.9	1.8	100	17.7	4.4	100
	network (double-sampling)	4.7	1.1	100	9.3	2.2	100	25.3	6.2	100
600	coldstart	422.3	15.6	76	494.2	17.3	52	540.1	18.3	40
	LPR	44.1	2.8	100	62.2	4.5	100	78.2	5.9	100
	column pre-population	37.9	1.1	100	39.1	1.2	100	42.8	1.5	100
	nearest neighbour	12.7	1.2	100	24.5	2.4	100	50.7	4.7	100
	random forest	12.3	1.2	100	29.3	2.9	100	55.8	5.3	100
	network (single-sampling)	11.4	1.1	100	19.8	1.9	100	39.1	3.7	100
	network (double-sampling)	12.7	1.1	100	21.1	1.8	100	37.0	3.1	100
1000	coldstart	450.3	13.3	84	507.4	14.9	73	544.4	16.1	59
	LPR	64.2	2.1	100	95.1	3.8	100	117.4	5.0	100
	column pre-population	60.6	1.4	100	67.7	1.8	100	76.1	2.2	100
	nearest neighbour	23.8	1.5	100	45.4	2.7	100	76.8	4.4	100
	random forest	21.4	1.3	100	50.1	3.0	100	83.8	4.8	100
	network (single-sampling)	19.2	1.2	100	35.5	2.2	100	62.4	3.7	100
	network (double-sampling)	18.6	1.0	100	26.4	1.4	100	48.1	2.5	100

Table 4: Performance of the column generation procedure with primal heuristics. The three columns labelled as “time”, “iters” and “solved” show average computational time (seconds) and iterations, and the number of problems solved within the time limit (problems where the column generation procedure found a near-optimal primal solution), respectively.

size	method	1% optimality			0.5% optimality			0.25% optimality		
		time	iters	solved	time	iters	solved	time	iters	solved
200	CPLEX	261.4	-	98	271.3	-	98	392.6	-	94
	coldstart	106.7	16.4	100	151.3	20.4	100	237.6	27.2	99
	LPR	8.3	1.1	100	18.8	3.4	100	60.8	11.6	100
	column pre-population	20.0	1.0	100	21.2	1.3	100	51.8	7.0	100
	nearest neighbour	4.8	1.0	100	8.3	1.7	100	34.2	6.8	100
	random forest	4.9	1.1	100	9.2	2.0	100	44.4	8.8	100
	network (single-sampling)	5.4	1.1	100	12.5	2.6	100	40.9	8.1	100
	network (double-sampling)	4.8	1.0	100	8.4	1.7	100	36.6	7.3	100
600	CPLEX	593.2	-	6	594.2	-	5	597.9	-	4
	coldstart	416.2	13.9	83	505.3	15.9	53	564.0	17.4	25
	LPR	29.1	1.1	100	44.9	2.2	100	93.5	5.7	100
	column pre-population	39.0	1.1	100	42.0	1.3	100	59.0	2.4	100
	nearest neighbour	12.7	1.0	100	21.8	1.6	100	51.5	3.9	100
	random forest	13.1	1.0	100	22.8	1.8	100	56.5	4.3	100
	network (single-sampling)	14.3	1.1	100	28.8	2.1	100	53.0	3.9	100
	network (double-sampling)	13.3	1.0	100	16.8	1.2	100	44.5	3.1	100
1000	CPLEX	600.0	-	0	600.0	-	0	600.0	-	0
	coldstart	508.9	11.8	63	559.5	13.1	36	590.0	14.0	15
	LPR	46.5	1.0	100	72.2	2.0	100	131.2	4.6	100
	column pre-population	61.9	1.3	100	74.3	1.8	100	112.6	3.4	100
	nearest neighbour	19.5	1.0	100	43.3	2.0	100	90.7	4.2	100
	random forest	20.8	1.1	100	43.5	2.0	100	98.0	4.4	100
	network (single-sampling)	18.9	1.0	100	49.5	2.3	100	94.0	4.4	100
	network (double-sampling)	20.1	1.0	100	24.9	1.2	100	65.9	2.9	100

Appendix B.

Table 4 shows the average computational time and the average number of iterations to close the optimality gap or to reach the time limit of 10 minutes, and the number of problems solved within the time limit. For comparison, we also solve the extensive form (2.1) to the same tolerances without decomposition (i.e., by branch and bound) using CPLEX.

These results show that solving the problems without decomposition is the slowest. The other methods show overall the same trend as Table 3. Solving the problem with coldstart is by far the slowest in every case. The column pre-population initialisation is slower than the remaining methods with the loose tolerance (1.0%), but faster than the LPR initialisation method with the tighter tolerance (0.5% or 0.25%). The methods based on machine learning are faster in all cases than the LPR, column pre-population and coldstart initialisation methods. When the problem is small or the tolerance is loose, the neural network with the double-sampling training is one of the best methods. On the large instances (600 or 1000 generators) with the tight tolerance (0.5% or 0.25%) it gives the best performance. We note that, as we saw in Table 3, there is again a correlation between the tightness of initial lower bounds computed by an initialisation method and the number of iterations required by the column generation procedure. For example,

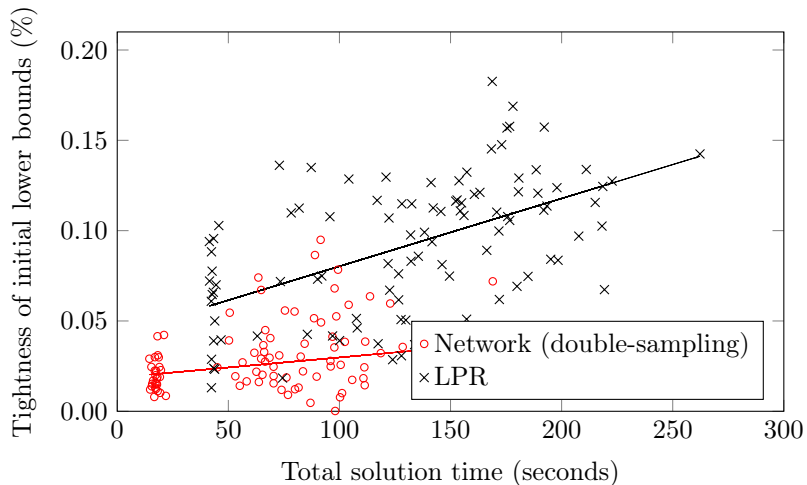


Figure 2: Tightness of the initial lower bound vs the total computational time required by the column generation procedure on 1000-generator test instances

comparing the pre-populate and the double sample results only 2 cases of 9 (the 1% and 0.5% for 600 generators) are against the trend and in these the deviation is minor.

To observe the effect of the quality of the initial dual values on the time taken by the column generation procedure more clearly, in Figure 2 the total computational time (0.25% tolerance) for each 1000-generator test instance is plotted against the tightness of the initial lower bound. The initialisation methods based on the LPR and the neural network model with the double-sampling training are compared in the plot. We observe a correlation between the two metrics. That is, if the lower bound computed in the first iteration of the column generation procedure is tighter then the total computational time required by the column generation procedure with primal heuristic tends to be smaller.

A method to generate initial dual values must balance the time to train, the quality of the dual values and the computational time. One extreme example is the LPR initialisation. It does not require any offline training. However, it produces dual values with a relatively loose lower bound and the solution time grows as the problem size increases. The methods based on machine learning models need offline training, but they run very quickly when solving a new instance and output dual values with a tight lower bound, resulting in a significant reduction of the time required to solve new instances.

To clarify this point, Table 5 shows a breakdown of the average computational time of the column generation procedures applied to the 1000-generator instances for the 100 test cases. The column labelled ‘Initialisation’ shows the time required to run the initialisation methods. For the LPR initialisation method this is the time to solve the LPR, while for the column pre-population initialisation method, this is the time to solve the unregularised RMP. For the methods based on machine learning, this is the time to evaluate the models. In the case of 1% tolerance, the time required to solve the LPR is longer than the sum of the time spent on the other routines. However, as the tolerance becomes tighter, the number of iterations and the time spent in the other routines increase, and by 0.25% tolerance, the LPR initialisation time is only 25% of the total time. We see a similar trend if we use the column pre-population initialisation.

Table 5: Breakdown of the average computational time (seconds) for 1000-generator case.

tolerance		Initialisation	RMP	Subproblem	Primal Heuristic
1%	coldstart	-	238.9	186.9	90.0
	LPR	29.6	0.1	12.2	4.7
	column pre-population	34.5	0.7	22.2	4.5
	nearest neighbour	0.0	0.1	15.0	4.4
	random forest	0.0	0.1	15.6	5.0
	network (single-sampling)	0.0	0.1	15.2	3.6
	network (double-sampling)	0.0	0.1	16.6	3.4
0.5%	coldstart	-	269.2	209.7	93.7
	LPR	29.6	1.0	28.9	12.8
	column pre-population	34.5	2.4	30.6	6.7
	nearest neighbour	0.0	0.7	31.1	11.5
	random forest	0.0	0.9	30.5	12.1
	network (single-sampling)	0.0	0.9	35.4	13.2
	network (double-sampling)	0.0	0.2	20.0	4.8
0.25%	coldstart	-	289.9	222.8	95.9
	LPR	29.6	4.8	69.0	27.8
	column pre-population	34.5	5.5	59.5	13.0
	nearest neighbour	0.0	2.8	65.1	22.8
	random forest	0.0	3.5	68.3	26.3
	network (single-sampling)	0.0	2.9	67.7	23.4
	network (double-sampling)	0.0	1.2	50.0	14.6

5 Conclusion

We have investigated the use of machine learning techniques to accelerate Dantzig-Wolfe decomposition with a column generation procedure to solve parametrized UC problems. In particular, we have proposed the use of machine learning models to compute the initial dual values for the column generation procedure. We have trained machine learning models so that they output dual values that yield tight dual lower bounds. In contrast to previous approaches which construct primal solutions directly, our approach can deliver an accurate solution as well as obtain its suboptimality bound, from the column generation procedure.

We have considered two approaches to training a machine learning model. The first approach, the single-sampling training, was to directly predict the optimal dual values given the problem parameter values as a standard regression problem. This requires a large dataset of the optimal dual values of training instances. The second approach, the double-sampling training, was based on the decomposable structure of the dual function.

We have first evaluated the performance of the machine learning models by computing the dual lower bounds at the dual values generated by the models. As benchmarks, we have used the dual lower bounds using the optimal dual values to the LPR, the optimal dual values to the RMP with pre-populated columns and coldstart with dual values $y = 0$. The coldstart initialisation always has the worst dual lower bounds, while the LPR generates dual values with relatively tight dual lower bounds. The column pre-population initialisation gives the best dual lower bounds on 200-generator instances, but the performance is poorer on the largest test instances. The methods based on machine learning yield dual values with significantly tighter dual lower

bounds, compared to the bound computed by the LPR, and the best bounds on the larger test instances. The computational time to solve the LPR or the RMP with pre-populated columns grows significantly as the instance size becomes larger. However, the computational time to evaluate the machine learning models remains negligible even with large instances.

To see the effectiveness of the above initialisation methods to warmstart the column generation procedure, we have evaluated the performance of the column generation procedure when warmstarted with the dual values obtained by each initialisation method. For this reason, the column generation procedure was run without any primal heuristics. In this evaluation, the time required by the column generation procedure to find near-optimal dual lower bounds was measured. The results revealed that the column generation procedure initialised with any of all the machine learning based initialisation methods can find a near-optimal dual lower bound more quickly than with the LPR, column pre-population or coldstart initialisation.

We have also evaluated the performance of the warmstarted column generation procedure in a more practical setup where a near-optimal primal solution (generator schedule) is sought. In this experiment, to find primal feasible solutions, the column generation procedure was run with primal heuristics and we observed the time required to find a primal solution of a prescribed suboptimality. In the numerical experiments, we observed that solving the UC problem with decomposition was always faster than solving the problem without decomposition using CPLEX. We further noted that warmstarting the column generation procedure successfully reduced the number of iterations and overall computational time to find a solution of prescribed suboptimality and this was especially dramatic for the initialisation methods based on machine learning. We observed that the initialisation methods that generate tighter initial Lagrangian lower bounds produce a better performance of the column generation procedure with primal heuristics. For example, the initialisation methods based on machine learning outperformed the LPR for all problem sizes. The numerical experiments also showed that the methods based on machine learning scale well and can be effective for solving large-scale UC problems. In particular, the neural network initialisation with the double-sampling training was usually the best, especially on the large instances (Table 2). One possible reason for the strong performance of the neural network with the double-sampling training compared to the other machine learning models is the efficiency of the training. The double-sampling training uses a stochastic gradient method which exploits the structure of the problem. With the training time we used (24 hours), we do not observe a significant difference between the methods on 200-generator instances, however, on larger instances, there were differences between them.

An interesting area for further study is the UC problem with stochastic demand forecasts. In many solution methods, UC problems with deterministic demand forecasts appear as subproblems and are solved repeatedly. See Takriti et al. (1996) and Schulze et al. (2017) for instance. Typically the subproblems share the same problem structure and only differ in the cost coefficients and the demand forecasts. This paper considers only the case where the perturbations are on the demand but not on the cost. However, the technique we developed is extendable to the stochastic case and may be equally effective.

A Problem Formulation

We closely follow one of the standard formulations in literature, referred to as the 3-binary variable formulation by Ostrowski et al. (2012), and formulate the following constraints:

- **Load balance:** Generators have to meet all the demand in each time period (generation shedding at 0 cost is allowed).
- **Reserve:** To deal with contingencies, it is required to keep a sufficient amount of backup in each time period, which can be activated quickly.
- **Power output bounds:** Each generator's power output has to be within its limit.
- **Ramp rate bounds:** Generators can only change their outputs within the ramp rates.
- **Minimum up/downtime:** If switched on (off), each generator has to stay on (off) for a given minimum period.

The formulation of the model is as follows.

- Parameters

- G : number of generators
- T : number of time periods where decisions are taken
- C_g^{nl} : no-load cost of generator g
- C_g^{mr} : marginal cost of generator g
- C_g^{up} : startup cost of generator g
- $P_g^{\text{max/min}}$: maximum/minimum generation limit of generator g
- $P_g^{\text{ru/rd}}$: operating ramp up/down limits of generator g
- $P_g^{\text{su/sd}}$: startup/shutdown ramp limits of generator g
- $T_g^{\text{u/d}}$: minimum up/downtime of generator g
- P_t^{d} : power demand at time t
- P_t^{r} : reserve requirement at time t

- Variables

- $\alpha_{gt} \in \{0, 1\}$: 1 if generator g is on in period t , and 0 otherwise
- $\gamma_{gt} \in \{0, 1\}$: 1 if generator g starts up in period t , and 0 otherwise
- $\eta_{gt} \in \{0, 1\}$: 1 if generator g shuts down in period t , and 0 otherwise
- $p_{gt} \geq 0$: power output of generator g in period t

- Total cost (the objective to be minimised)

$$\min \sum_{t=1}^T \sum_{g=1}^G \left(C_g^{\text{nl}} \alpha_{gt} + C_g^{\text{mr}} p_{gt} + C_g^{\text{up}} \gamma_{gt} \right).$$

- Load balance

$$\sum_{g=1}^G p_{gt} \geq P_t^{\text{d}} \quad t = 1, 2, \dots, T.$$

- Reserve

$$\sum_{g=1}^G (P_g^{\max} \alpha_{gt} - p_{gt}) \geq P_t^r \quad t = 1, 2, \dots, T.$$

- Power output bounds

$$P_g^{\min} \alpha_{gt} \leq p_{gt} \leq P_g^{\max} \alpha_{gt} \quad g = 1, 2, \dots, G, t = 1, 2, \dots, T$$

- Ramp rate bounds

$$p_{gt} - p_{gt-1} \leq P_g^{\text{ru}} \alpha_{gt-1} + P_g^{\text{su}} \gamma_{gt} \quad g = 1, 2, \dots, G, t = 2, 3, \dots, T.$$

$$p_{gt-1} - p_{gt} \leq P_g^{\text{rd}} \alpha_{gt} + P_g^{\text{sd}} \eta_{gt} \quad g = 1, 2, \dots, G, t = 2, 3, \dots, T.$$

- Minimum up/downtime

$$\sum_{u=\max\{t-T_g^u+1, 1\}}^t \gamma_{gu} \leq \alpha_{gt} \quad g = 1, 2, \dots, G, t = 1, 2, \dots, T$$

$$\sum_{u=\max\{t-T_g^d+1, 1\}}^t \eta_{gu} \leq 1 - \alpha_{gt} \quad g = 1, 2, \dots, G, t = 1, 2, \dots, T$$

- Logical constraints (to enforce binaries to work as we expect)

$$\alpha_{gt} - \alpha_{gt-1} = \gamma_{gt} - \eta_{gt} \quad g = 1, 2, \dots, G, t = 2, 3, \dots, T$$

$$1 \geq \gamma_{gt} + \eta_{gt} \quad g = 1, 2, \dots, G, t = 1, 2, \dots, T$$

B Implementation Details

In the numerical experiments, a regularised column generation procedure is used. Initially, the regularisation centre is set to the dual values given by the initialisation method. In each iteration, a lower bound is evaluated using the current dual values and the regularisation centre is updated to the current dual values if the lower bound gets improved. Furthermore, if the lower bound improves, the regularisation parameter is divided by two, and otherwise multiplied by two. This closely follows the implementation of the column generation procedure described by Schulze et al. (2017).

In every iteration, after the pricing subproblems are solved, a primal heuristic based on local search is run. Given the solutions to the pricing subproblems, this primal heuristic checks the feasibility and if necessary switches on the cheapest available generators to make the solution feasible. Note that the solutions to the pricing subproblems are feasible generator schedules and infeasibility only arises from an insufficient generation capacity to meet the demand or insufficient reserve. This primal heuristic loosely follows that proposed by Guan et al. (1992). In our experiments, we observe that it usually finds near-optimal primal solutions when the dual values get close to optimal. If the column generation procedure using the local search primal heuristic fails to find a primal feasible solution satisfying the given optimality tolerance within

30 iterations, we use in addition a column combination primal heuristic. Then in the subsequent iterations, we use both of the primal heuristics (the two primal heuristics are independent of each other). The column combination primal heuristic is a popular primal heuristic in the column generation procedure in general. The idea of this heuristics is to solve (2.1) with restricted patterns of solutions, referred to as restricted master IP by Vanderbeck (2005). In the k th iteration, we replace X_s in (2.1) with $\{x_s^{(k-l)}\}_{l=0,1,2}$ where $\{x_s^{(k-l)}\}_{l=0,1,2}$ are solutions to the s th pricing subproblem found in the current iteration and or the previous two iterations. The resulting problem is still a mixed-integer programme but the solution space is much smaller than the original problem. In our numerical experiments, we observed that the problem is feasible without the need to add artificial variables (columns).

C Hyperparameters for the Neural Networks

The model consists of 4 hidden layers of 1000 units per layer, with skip connections between hidden layers as described by De and Smith (2020). The structure of the network is visualised in Figure 3. The tanh function is used as an activation function. These hyperparameters were chosen by grid search, as discussed in the end of this section. The weights in the linear transformation are initialised based on the methods of Glorot and Bengio (2010). All biases and the weights on the residual connections are initialised to zero.

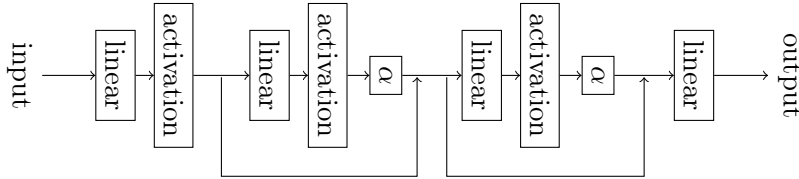


Figure 3: The structure of the neural network model with skip connections. For a more detailed explanation of the architecture, see De and Smith (2020).

For the single-sampling training, we use the dataset of the optimal dual values of training instances. We split the dataset into a training set (80%) and a validation set (20%). The Adam method (Kingma and Ba (2015)) is used to learn the parameters of the neural network. After each epoch, we evaluate the mean squared error on the validation set. If it does not improve for successive 4 epochs, we halve the learning rate. If it does not improve for successive 12 epochs, we terminate the training to avoid overfitting.

For the double-sampling training, we also use the Adam method (Kingma and Ba (2015)). The model performance is evaluated every 5 minutes by computing the dual lower bound on validation instances using the output of the neural network. To this end, 10 validation instances are sampled, which are distinct from the instances used to train the model (i.e. compute (3.2)) and to test the final performance (reported in Table 3, 4). If it fails to improve the performance for successive 15 minutes, the learning rate is divided by 1.5. In our experiment, we did not observe overfitting. However, if this became an issue, we could use regularisation such as dropout, as proposed by Cobbe et al. (2019).

As discussed earlier in this section, the number of layers and the activation function were chosen by grid search. Models with 3, 4 and 5 layers and with tanh and relu were trained on the 200-generator case. Table 6 reports the performance of the trained models in the same format

Table 6: Tightness of lower bound computed with the output of the neural networks. The tightness is measured by comparing it with the best known lower bound lb_{CPLEX} obtained by running CPLEX for 2 hours and is reported as the percentage of lb^* .

activation	number of layers	$\text{lb}_{\text{CPLEX}} - \text{lb}_1$
tanh	3	0.0239
	4	0.0228
	5	0.0239
relu	3	0.0462
	4	0.0480
	5	0.0479

Table 7: Difference of primal and dual values between training and test instances.

size	difference in terms of primal values (%)	difference in terms of dual values (%)
200	2.93	6.85
600	2.88	5.99
1000	2.80	7.09

as Table 2. The instances used in Table 6 are different from the training, validation (used to monitor the progress of the training) and test instances.

D Statistics Relevant to Training

Table 7 shows the difference between training and test instances. For each test instance the commitment decision of the nearest training instance p^{train} and the commitment decision of the test instance p^{test} was compared using the Hamming distance:

$$\text{Difference in terms of primal values (\%)} = \frac{\text{Hamming distance of } p^{\text{train}} \text{ and } p^{\text{test}}}{\text{the number of commitment decisions in } p^{\text{test}}} \cdot 100.$$

The average of the above quantity using 40 test instances was computed and shown in the middle column of Table 7. Similarly, the distance between the optimal dual values of the nearest training instances and the test instances was computed. For each test instance, the nearest training instance is obtained. Then, the optimal dual values of the training instance and the test instance (d^{train} and d^{test} , respectively) and the following metric is computed:

$$\text{Difference in terms of dual values (\%)} = \frac{\|d^{\text{train}} - d^{\text{test}}\|_2}{\|d^{\text{test}}\|_2} \cdot 100.$$

The average is shown in the right-most column in the table.

References

- Bard, J. F. (Sept. 1988). “Short-term scheduling of thermal-electric generators using Lagrangian relaxation”. In: *Operations Research* 36.5, pp. 756–766. DOI: <https://doi.org/10.1287/opre.36.5.756>.

- Bengio, Y., A. Lodi, and A. Prouvost (2021). “Machine learning for combinatorial optimization: a methodological tour d’horizon”. In: *European Journal of Operational Research* 290.2, pp. 405–421. ISSN: 0377-2217. DOI: <https://doi.org/10.1016/j.ejor.2020.07.063>.
- Bertsekas, D. et al. (1983). “Optimal short-term scheduling of large-scale power systems”. In: *IEEE Transactions on Automatic Control* 28.1, pp. 1–11. ISSN: 0018-9286.
- Bertsekas, D. P. (2009). *Convex optimization theory*. Belmont, Massachusetts: Athena Scientific. ISBN: 1886529310.
- Borghetti, A. et al. (2002). “Lagrangian heuristics based on disaggregated bundle methods for hydrothermal unit commitment”. In: *IEEE Power Engineering Review* 22.12, pp. 60–60. ISSN: 0272-1724. DOI: <https://doi.org/10.1109/TPWRS.2002.807114>.
- Briant, O. et al. (2008). “Comparison of bundle and classical column generation”. In: *Mathematical Programming* 113.2, pp. 299–344. ISSN: 0025-5610. DOI: <https://doi.org/10.1007/s10107-006-0079-z>.
- Cobbe, K. et al. (Sept. 2019). “Quantifying generalization in reinforcement learning”. In: *Proceedings of the 36th international conference on machine learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 1282–1289. URL: <https://proceedings.mlr.press/v97/cobbe19a.html>.
- Dalal, G. et al. (2018). “Unit commitment using nearest neighbor as a short-term proxy”. In: *2018 power systems computation conference*, pp. 1–7. DOI: [10.23919/PSCC.2018.8442516](https://doi.org/10.23919/PSCC.2018.8442516).
- De, S. and S. Smith (2020). “Batch normalization biases residual blocks towards the identity function in deep networks”. In: *Advances in neural information processing systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 19964–19975. URL: <https://proceedings.neurips.cc/paper/2020/hash/d5ade38a2c9f6f073d69e1bc6b6e64c1-Abstract.html>.
- Gasse, M. et al. (2019). “Exact combinatorial optimization with graph convolutional neural networks”. In: *Advances in neural information processing systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 15580–15592.
- Glorot, X. and Y. Bengio (Jan. 2010). “Understanding the difficulty of training deep feedforward neural networks”. In: *Journal of Machine Learning Research - Proceedings Track* 9, pp. 249–256. URL: <http://proceedings.mlr.press/v9/glorot10a.html>.
- Guan, X. et al. (1992). “An optimization-based method for unit commitment”. In: *International Journal of Electrical Power & Energy Systems* 14.1, pp. 9–17. ISSN: 0142-0615. DOI: [https://doi.org/10.1016/0142-0615\(92\)90003-R](https://doi.org/10.1016/0142-0615(92)90003-R).
- Guha, N. et al. (2019). “Machine Learning for AC Optimal Power Flow”. In: *The 36th International Conference on Machine Learning. Climate Change: How Can AI Help?*
- Hiriart-Urruty, J. and C. Lemaréchal (1993). *Convex analysis and minimization algorithms ii*. Grundlehren der mathematischen Wissenschaften 306. Berlin: Springer. ISBN: 978-3-642-08162-0. DOI: [10.1007/978-3-662-06409-2](https://doi.org/10.1007/978-3-662-06409-2).
- Hutter, F., H. H. Hoos, and K. Leyton-Brown (2011). “Sequential model-based optimization for general algorithm configuration”. In: vol. 6683, pp. 507–523. ISBN: 9783642255656. DOI: https://doi.org/10.1007/978-3-642-25566-3_40.
- Jones, D. R. (2001). “A taxonomy of global optimization methods based on response surfaces”. In: *Journal of Global Optimization* 21, pp. 345–383. DOI: <https://doi.org/10.1023/A:1012771025575>.

- Khalil, E. et al. (Feb. 2016). “Learning to branch in mixed integer programming”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1. DOI: <https://doi.org/10.1609/aaai.v30i1.10080>. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/10080>.
- Kingma, D. P. and J. Ba (May 2015). “Adam: a method for stochastic optimization”. In: *The international conference on learning representations*. URL <https://arxiv.org/abs/1606.01885>.
- Lodi, A. and G. Zarpellon (2017). “On learning and branching: a survey”. In: *TOP* 25.2, pp. 207–236. ISSN: 1134-5764. DOI: <https://doi.org/10.1007/s11750-017-0451-6>.
- Morabit, M., G. Desaulniers, and A. Lodi (2021). “Machine-learning-based column selection for column generation”. In: *Transportation Science* 55.4, pp. 815–831. DOI: [10.1287/trsc.2021.1045](https://doi.org/10.1287/trsc.2021.1045). eprint: <https://doi.org/10.1287/trsc.2021.1045>. URL: <https://doi.org/10.1287/trsc.2021.1045>.
- Nair, V. et al. (2018). “Learning fast optimizers for contextual stochastic integer programs”. In: *Proceedings of the thirty-fourth conference on uncertainty in artificial intelligence, UAI 2018, monterey, california, usa, august 6-10, 2018*, pp. 591–600. URL: <http://auai.org/uai2018/proceedings/papers/217.pdf>.
- Ostrowski, J., M. F. Anjos, and A. Vannelli (2012). “Tight mixed integer linear programming formulations for the unit commitment problem”. In: *IEEE Transactions on Power Systems* 27.1, pp. 39–46. ISSN: 0885-8950. DOI: <https://doi.org/10.1109/TPWRS.2011.2162008>.
- Owerko, D., F. Gama, and A. Ribeiro (2020). “Optimal power flow using graph neural networks”. In: *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5930–5934. DOI: <https://doi.org/10.1109/ICASSP40776.2020.9053140>.
- Pineda, S. and J. M. Morales (2022). “Is learning for the unit commitment problem a low-hanging fruit?” In: *Electric Power Systems Research* 207, p. 107851. ISSN: 0378-7796. DOI: <https://doi.org/10.1016/j.epsr.2022.107851>. URL: <https://www.sciencedirect.com/science/article/pii/S0378779622000815>.
- Schulze, T., A. Grothey, and K. McKinnon (Aug. 2017). “A stabilised scenario decomposition algorithm applied to stochastic unit commitment problems”. In: *European Journal of Operational Research* 261.1, pp. 247–259. ISSN: 0377-2217. DOI: <https://doi.org/10.1016/j.ejor.2017.02.005>.
- Shen, Y. et al. (June 2022). “Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.9, pp. 9926–9934. DOI: [10.1609/aaai.v36i9.21230](https://doi.org/10.1609/aaai.v36i9.21230). URL: <https://ojs.aaai.org/index.php/AAAI/article/view/21230>.
- Takriti, S., J. R. Birge, and E. Long (1996). “A stochastic model for the unit commitment problem”. In: *IEEE Transactions on Power Systems* 11.3, pp. 1497–1508. ISSN: 0885-8950. DOI: <https://doi.org/10.1109/59.535691>.
- Václavík, R. et al. (2018). “Accelerating the branch-and-price algorithm using machine learning”. In: *European Journal of Operational Research* 271.3, pp. 1055–1069. ISSN: 0377-2217.
- van Ackooij, W. et al. (2018). “Large-scale unit commitment under uncertainty: an updated literature survey”. In: *Annals of Operations Research* 271.1, pp. 11–85. ISSN: 0254-5330.
- van Hoai, T., G. Reinelt, and H. G. Bock (2005). “Advanced column generation techniques for crew pairing problems”. In: *Modeling, simulation and optimization of complex processes*. Ed. by H. G. Bock et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 203–214. ISBN: 978-3-540-27170-3.

- Vanderbeck, F. (2005). “Implementing mixed integer column generation”. In: *Column generation*. Springer US, pp. 331–358. ISBN: 0387254854.
- Vanderbeck, F. and M. W. P. Savelsbergh (2006). “A generic view of Dantzig–Wolfe decomposition in mixed integer programming”. In: *Operations Research Letters* 34.3, pp. 296–306. ISSN: 0167-6377.
- Xavier, A. S., F. Qiu, and S. Ahmed (2020). “Learning to solve large-scale security-constrained unit commitment problems”. In: *INFORMS Journal on Computing* 0.0. DOI: <https://doi.org/10.1287/ijoc.2020.0976>.
- Yang, Y. and L. Wu (2021). “Machine learning approaches to the unit commitment problem: current trends, emerging challenges, and new strategies”. In: *The Electricity Journal* 34.1. Special Issue: Machine Learning Applications To Power System Planning And Operation, p. 106889. ISSN: 1040-6190. DOI: <https://doi.org/10.1016/j.tej.2020.106889>. URL: <https://www.sciencedirect.com/science/article/pii/S1040619020301810>.
- Zamzam, A. S. and K. Baker (2020). “Learning optimal solutions for extremely fast ac optimal power flow”. In: *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 1–6. DOI: <https://doi.org/10.1109/SmartGridComm47815.2020.9303008>.