Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the manuscript number!)

Use of a neural network to warmstart column generation for unit commitment

Nagisa Sugishita, Andreas Grothey, Ken McKinnon
The University of Edinburgh, n.sugishita@sms.ed.ac.uk, a.grothey@ed.ac.uk, k.mckinnon@ed.ac.uk, https://www.maths.ed.ac.uk/

This paper focuses on solution methods for unit commitment problems which are to be solved repeatedly with different data but with the same problem structure. Dantzig-Wolfe decomposition with a column generation procedure has been shown to be successful for the unit commitment problem. We demonstrate use of a neural network to generate initial dual values for the column generation procedure, and show how the neural network can be trained efficiently using dual decomposition. Our numerical experiments compare our methods with baselines based on the linear programming relaxation and coldstart as well as other machine learning based approaches: random forest and nearest neighbour. They reveal that the machine learning approaches, in particular the one based on a neural network, solves test instances to high precision in shorter computational time and scales to handle larger-scale instances.

Key words: unit commitment, Dantzig-Wolfe decomposition, column generation, warmstarting, machine learning
History:

1. Introduction

The unit commitment (UC) problem is an important optimisation problem in the energy industry. It aims to compute the optimal operating schedules of power plants 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. Given a demand forecast over a planning horizon, the timing of switching and the amount of power dispatch have to be optimised simultaneously. This gives rise to large-scale combinatorial problems and 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 which 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 and problems are solved with different demand forecasts. This allows us to use machine learning techniques to accelerate solution methods.

One popular approach for UC problems is to use Dantzig-Wolfe decomposition to decompose the problem by generators. 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 discussed by Briant et al. (2008). When the problem at hand is a mixed-integer linear programming (MILP) problem, the reformulation is not exact but a relaxation of the original problem. However Bertsekas et al. (1983) and Bard (1988) reported that the optimality gap introduced by the reformulation is typically small especially if the problem size is large. In such cases, Dantzig-Wolfe decomposition with a suitable primal heuristic is likely to solve the UC problem to high precision.

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

To warmstart the column generation procedure we use a pre-trained model to generate initial dual values. The training of the neural network is based on the method proposed by Nair et al. (2018). In their work, neural networks were trained to compute dual values to yield strong lower bounds on integer two-stage stochastic programming. In our work, we combine the neural network with the column generation procedure to solve UC and obtain guaranteed lower and upper bounds on the optimal objective value. After the training, when solving a new instance, the neural network is used with problem data as input to generate dual values that yield tight initial lower bounds. 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 obtain feasible solutions. With this approach, we can exploit the strength of the neural network while maintaining the desirable property of the solution method, such as exactness. We have tested the above approach on large-scale UC problems.

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

2. Dantzig-Wolfe decomposition and column generation

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 mixed-integer programmes parametrized by ω:

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

where $x_1, x_2, \ldots, x_G$ are vectors of decision variables and

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

In our model $X_g(\omega)$ is non-empty and bounded for every ω and $g = 1, 2, \ldots, G$, and that the problem (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.

In Dantzig-Wolfe decomposition we consider a relaxation of (1), referred to as the master problem (MP), by replacing X_g with $\text{conv}(X_g)$ for every g. Let $\{x_{gi} \mid i \in I_g\}$ denote the extreme points of X_g. Given the boundedness assumption on X_g, the MP can be written as

$$\min_p \sum_{g=1}^{G} \sum_{i \in I_g} {c_g}^T x_{gi} p_{gi} \qquad (2)$$

$$\begin{aligned}
\text{s.t.} \ & \sum_{g=1}^{G} \sum_{i \in I_g} A_g x_{gi} p_{gi} = a, \\
& \sum_{i \in I_g} p_{gi} = 1, \quad g = 1, 2, \ldots, G, \\
& p_{gi} \geq 0, \quad g = 1, 2, \ldots, G, i \in I_g.
\end{aligned}$$

Since the MP is too large to formulate and solve explicitly, a column generation procedure is used. The restricted master problem (RMP) is defined by replacing I_g in the MP with a subset $\hat{I}_g \subset I_g$ for every g. Suppose that the RMP is feasible ($\hat{I}_g$'s must have suitable columns to satisfy the first constraint in (2)) and let y and σ_g for $g = 1, 2, \ldots, G$ be the optimal dual solution to the RMP corresponding to the first and second constraint respectively. To find columns to be added to the RMP, the following subproblems, known as the pricing subproblems, are solved:

$$r_g(y) = \min_{x_g} \{ ({c_g}^T - y^T A_g) x_g \mid x_g \in X_g \}, \quad g = 1, 2, \ldots, G. \tag{3}$$

If $r_g(y) \geq \sigma_g$ for all g, the RMP has found the optimal solution to the MP. Otherwise, the solutions to the pricing subproblems are added to the RMP and the above process is repeated. It follows from linear programming (LP) duality that given dual values y

$$q(y) = a^T y + \sum_{g=1}^{G} r_g(y) \tag{4}$$

is a lower bound to the MP, and since the MP is a relaxation of (1) it is also a lower bound on z.

It is known that the naive column generation approach described above suffers from instability. To tackle the issue quadratic regularisation of the dual variables may be added, as is described by Briant et al. (2008). Dualizing the RMP and adding quadratic regularisation on the dual variables gives

$$\begin{aligned}
\max_{y,\sigma} \ & a^T y + \sum_{g=1}^{G} \sigma_g - \frac{\mu}{2} \| y - \bar{y} \|_2^2 \qquad (5) \\
\text{s.t.} \ & \sigma_g \leq ({c_g}^T - y^T A_g) x_{gi} \quad g = 1, 2, \ldots, G, i \in \hat{I}_g, \\
& y, \sigma : \text{free},
\end{aligned}$$

where $\bar{y}$ is a regularisation centre and μ is a parameter to adjust the strength of the regularisation. In the regularised column generation procedure, the regularised RMP is used in place of the RMP. The optimal solution to (5) 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 when the lower bound (4) has improved.

3. Warmstarting

In the above algorithm, the dual variables 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 the dual variables y. It follows that in the case where near-optimal dual values are used as an initial point the algorithm would provide a near-optimal lower bound in the first iteration. 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 expected that feeding near-optimal dual values as initial dual values help the algorithm to terminate in a shorter execution time.

It is worth noting that regularisation on the dual variables is crucial in this context. Without regularisation, even if near-optimal dual values are used as an 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 a further discussion.

As mentioned in the introduction Section 1, one option to find good dual values is to solve some approximation of (1) such as the linear programming relaxation (LPR) and obtain its optimal dual solution as used by Borghetti et al. (2002), Schulze et al. (2017). This approach does not require any offline computation or pre-trained models. On the other hand solving the approximations requires time.

An alternative approach, which we explore below, is to train a model which takes ω as input and outputs dual values y that provides a tight lower bound. The training approach is based on the work by Nair et al. (2018) where dual decomposition was applied to solve a parametrized two-stage stochastic programming problem.

Consider a neural network $f(\omega, \theta)$ which maps problem data ω and model parameter θ to y. We aim to learn values of the parameter θ so that given ω the model outputs dual values $y = f(\omega, \theta)$ for which the lower bound $q(y)$ in (4) is tight. In this section we explicitly state the dependency of the lower bound on ω as $q(\omega, y)$, which was suppressed in (4). 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

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

If subproblem g is sampled uniformly from $1, 2, \ldots, G$ from (4) it follows that

$$q(\omega, y) = a^T y + \sum_{g=1}^{G} r_g(y) = \frac{1}{|G|}\sum_{g=1}^{G}(a^T y + |G| r_g(y)) = \mathrm{E}_g[\widetilde{q}(\omega, y, g)],$$

where

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

Therefore we get

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

We follow the standard approach used for training a neural network which is to use the stochastic gradient ascent method. That is, we sample ω and g, compute the gradient of $\widetilde{q}$ with respect to θ and make a single gradient ascent step. We then resample ω and g and repeat the process.

The gradient of $\widetilde{q}$ with respect to θ can be computed as follows. Fix problem data ω and subproblem index g and compute the duals value $y = f(\omega, \theta)$ and the component $\widetilde{q}(\omega, y, g)$ of the 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 (3) are differentiable with respect to y. Then the gradient of $\widetilde{q}(\omega, y, g)$ with respect to y is given by

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

where ${x_g}^*$ is the solution to the pricing subproblem g (3). Using the chain rule, we obtain

$$\frac{\partial \widetilde{q}}{\partial \theta} = J \frac{\partial \widetilde{q}}{\partial y}, \tag{6}$$

where J is the Jacobian matrix of the neural network output $y = f(\omega, \theta)$ with respect to θ, which is typically given by automatic differentiation.

It is important to note that evaluating the above quantity (6) is much faster than solving the original problem or the MP to optimality. This is because it requires the solution of only a single pricing subproblem, which is typically substantially smaller than the original problem.

Once a model is trained, it can be used to compute initial dual values for the regularised column generation procedure. It is expected that unlike solving an approximation such as the LPR this model will run quickly and we later demonstrate that it produces near-optimal initial dual values.

4. Numerical experiments

In this section the proposed initialisation method based on a neural network is compared with standard approaches for the UC problem. All methods are implemented in Python. IBM ILOG CPLEX 20.1.0[1] is used as the optimisation solver (the barrier method for the RMP and the branch and bound method for the pricing subproblems) and PyTorch to implement the neural networks. The experiments are done on a workstation with 16 cores Intel® Xeon® E5-2670 with 126 GB of RAM.

4.1. Problem

In the experiments, we consider a setup in which UC problems are solved repeatedly with a fixed set of generators but different demand forecasts. To assess the scalability, we consider 3 different problem sizes; 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 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.[2] A detailed description of the problem formulation is given in Appendix A.

Dantzig-Wolfe decomposition with the regularised column generation procedure is used to solve the UC problems. Various initialisation methods are compared, and these are described in the next section. In each iteration of the column generation procedure, a local search primal heuristic is run to find a primal feasible solution. Implementation details including the primal heuristics are given in Appendix B.

[1] https://www.ibm.com/products/ilog-cplex-optimization-studio

[2] https://www.nationalgrideso.com/

4.2. Initialisation methods

We consider the following 5 methods to initialise the dual variables.

Neural Network: An initialisation method based on a neural network is implemented as described in Section 3. The neural network consists of 4 hidden layers of 1000 units per layer, with skip connections between hidden layers. For each set of generators, a single neural network is trained for 24 hours using 8 CPU cores. We follow the procedure used by Nair et al. (2018), except that only a single neural network is trained for each set of generators to reduce the computational load. The hyperparameters on the model architecture and the training are found in Appendix C.

Random forest: As a baseline of a trainable model, a method based on a random forest (Briant et al. (2008)) is also evaluated. On the same training budget (24 hours on 8 CPU cores), as many training instances as possible are solved to 0.25% optimality using Dantzig-Wolfe decomposition with the LPR initialisation and the local search primal heuristic. The parameter values and the corresponding optimal dual values of the training instances are saved. After the timelimit, a random forest is trained by supervised learning to predict the optimal dual values given a parameter value. The training took less than 30 seconds in all cases. Unlike our neural network approach, this approach requires the solution of many UC instances.

Nearest Neighbour: A nearest neighbour approach is also evaluated to provide an alternative trainable baseline. We use the dataset that was created to train the random forest model. At runtime, given a test instance, its parameter is compared against those of the training instances. The training instance with the closest parameter value in terms of the Euclidean distance is selected, and the corresponding optimal dual values are used as initial dual values. As in the random forest approach, this requires the solution of many UC instances.

LPR: This is the method based on the LPR described in Section 3. Given a test instance, we first solve the LPR by CPLEX. This method does not require any training. This is the method used by Schulze et al. (2017).

Coldstart: As a naive base line, column generation is run from $y = 0$. This method does not require any training.

Table 1 **Tightness of initial lower bounds lb_1 (%) and computational time (s). The tightness of initial lower bounds lb_1 is measured by comparing it with the best known lower bound lb^* obtained by running CPLEX for 4 hours and is reported as percentages.**

	size: 200		600		1000	
method	$\text{lb}^* - \text{lb}_1$	time	$\text{lb}^* - \text{lb}_1$	time	$\text{lb}^* - \text{lb}_1$	time
network	0.059	<0.1	0.051	<0.1	0.040	<0.1
forest	0.060	<0.1	0.073	<0.1	0.076	<0.1
nearest	0.048	<0.1	0.060	<0.1	0.063	<0.1
LPR	0.126	4.6	0.108	17.7	0.105	29.2
coldstart	99.854	0.0	99.831	0.0	99.873	0.0
LPR alone	0.179	4.6	0.176	17.7	0.163	29.2

4.3. Evaluation

To test the performances of the above methods, 40 test instances of each size were created and solved by each method. The demand data to construct the test instances was sampled from a different year than those of the training instances. Each method was run sequentially on a single CPU core until a solution within 1%, 0.5% and 0.25% suboptimality was found or the time limit of 10 minutes was reached.

Table 1 shows the tightness of the initial lower bound obtained in the first iteration of the column generation procedure and the required time (all times in this section are wall clock times) to run the initialisation methods. For each test instance, a lower bound on the optimal objective value was also computed by running CPLEX for 4 hours. Then the initial lower bounds obtained from the column generation procedure were compared with the best lower bound found by CPLEX. For all the test instances the lower bounds found by CPLEX after 4 hours were better than the initial lower bounds from the column generation procedure. In the table the average gaps of the bounds are reported. For comparison, the objective value of the LPR (which is also a valid lower bound) is shown in the table as well (labelled as LPR alone).

Clearly, coldstart with $y = 0$ yields poor lower bounds. Although the lower bounds of the remaining 4 methods are comparable, the lower bounds obtained from the methods based on machine learning are significantly tighter than LPR. The solution time of the LPR grows and it takes significant time especially on larger problems. On the other hand, the computational time of the other methods remains small. Observe that the optimal LPR objective value gives a lower bound, but evaluating the Lagrangian using the solution to the LPR yields a tighter lower bound.

Table 2 shows the number of problems solved by the column generation procedure within the time limit of 10 minutes, the average computational time and the average number of

Table 2 Number of problems solved, average computational time (s) and iterations.

		1% optimality			0.5% optimality			0.25% optimality		
size	method	solved	time	iters	solved	time	iters	solved	time	iters
200	network	40	4.6	1.0	40	9.0	1.9	40	**37.5**	7.4
	forest	40	**4.3**	1.0	40	**8.0**	1.8	40	42.0	8.2
	neighbour	40	4.7	1.0	40	10.0	2.1	40	43.1	8.7
	LPR	40	7.8	1.0	40	15.9	3.0	40	53.6	10.8
	coldstart	40	103.8	16.6	40	161.1	21.7	40	235.6	27.4
	CPLEX	39	235.8	-	39	244.1	-	38	336.9	-
600	network	40	**13.4**	1.0	40	**20.5**	1.5	40	**48.4**	3.4
	forest	40	14.0	1.1	40	26.5	2.0	40	55.5	4.2
	neighbour	40	13.8	1.1	40	27.6	2.0	40	57.8	4.4
	LPR	40	28.7	1.0	40	43.0	2.0	40	86.9	5.4
	coldstart	36	413.4	13.8	16	528.2	16.6	9	576.3	17.7
	CPLEX	4	594.1	-	4	594.1	-	4	594.1	-
1000	network	40	21.5	1.0	40	**34.3**	1.5	40	**74.2**	3.2
	forest	40	**19.5**	1.0	40	37.1	1.8	40	89.7	4.3
	neighbour	40	23.5	1.2	40	46.5	2.1	40	96.5	4.4
	LPR	40	45.6	1.0	40	67.4	1.8	40	120.4	4.2
	coldstart	23	515.8	12.2	7	567.8	13.6	5	584.0	14.0
	CPLEX	1	599.0	-	1	599.0	-	1	599.0	-

iterations to close the optimality gap or to reach 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 timelimit is used. For comparison, we also solve the extensive form (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. Among the remaining 5 methods, which are based on the column generation procedure, solving the problem with coldstart is by far the slowest in every case. The 3 methods based on neural network, random forest and nearest neighbour are faster in all cases than the method based on the LPR. The nearest neighbour is the slowest in many cases. The neural network and random forest have a similar performance on the loose tolerances 1.0% and 0.5% but the neural network outperforms all the other methods on 0.25% tolerance.

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 1 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 a neural network and the LPR are compared in the plot. We observe a correlation between the two metrics. That is, if the

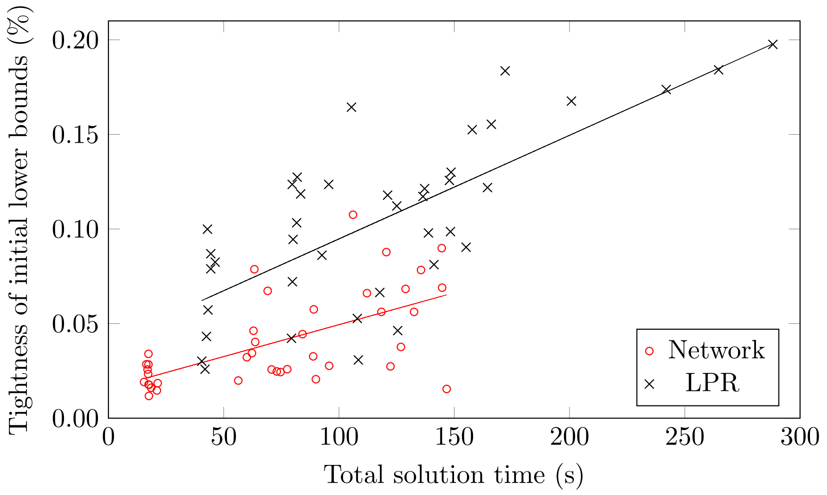

Figure 1 **Tightness of the initial lower bound vs the total computational time required by the column generation procedure**

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 tends to be smaller.

The method to generate initial dual values must balance the time to train, the quality of the dual values and the computational time on a new instance. One extreme example is the LPR. 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 method based on a neural network needs off-line training, but it runs very quickly when solving a new instance and outputs dual values with a tight lower bound, resulting in significant speed up of the time to solve new instances.

Table 3 shows a breakdown of the average computational time of the column generation procedures based on the neural network and on the LPR for the 1000-generator instances for the 40 test cases. The column labelled as ‘Initialisation’ shows the time required to run the initialisation methods. In the LPR-based method, this is the time to solve the LPR, while the method based on a neural network this is the time to evaluate the neural network (but does not include the time to train the neural network). In cases 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 increases, and by 0.25% tolerance the LPR initialisation time is only 25% of the total time.

Table 3 **Breakdown of the average computational time (s).**

tolerance	method	Initialisation	RMP	Subproblem	Primal Heuristic
1%	network	0.0	0.1	17.1	4.3
	LPR	29.2	0.1	11.8	4.5
0.5%	network	0.0	0.5	25.1	8.8
	LPR	29.2	0.8	25.2	12.2
0.25%	network	0.0	1.8	53.6	18.8
	LPR	29.2	3.8	61.4	26.1

5. Conclusion

We have considered a neural network to accelerate Dantzig-Wolfe decomposition with a column generation procedure to solve a parametrized UC problem. The neural network was first trained to generate values for the dual variables which yield tight Lagrangian lower bounds. The training was done efficiently using dual decomposition. We compared the Lagrangian lower bounds of this approach with the Lagrangian lower bounds obtained using the optimal dual solution to the LPR and the dual values generated by other supervised learning methods, namely random forest and nearest neighbour, and coldstart. The coldstart always has the worst Lagrangian lower bound. The LPR generates dual values with relatively tight Lagrangian lower bounds. However the computational time grows as the instance size became larger. The methods based on machine learning yields dual values with tighter Lagrangian lower bound than the LPR while the computational time remains negligible even with large instances. Among the methods based on machine learning, the neural network usually achieves the best Lagrangian lower bounds.

These methods were then used to warmstart the column generation procedure. 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. We observed that the initialisation methods that generates tighter initial Lagrangian lower bounds produce a better performance of the column generation procedure. For example, the initialisation methods based on machine learning outperformed the LPR in all cases and the neural network was usually the best among them, especially if the target tolerance was small. The numerical experiments also showed that the neural network approach scales well and can be effective for solving large-scale unit commitment problems.

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. In such cases, it would be of interest to use data gathered from previous instances and warmstart the algorithm when solving a new instance.

Acknowledgments

Appendix 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 back up 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
  — $P_g^{\mathrm{max}\,/\,\mathrm{min}}$: maximum/minimum generation limit of generator g
  — $P_g^{\mathrm{ru/rd}}$: operating ramp up/down limits of generator g
  — $P_g^{\mathrm{su/sd}}$: startup/shutdown ramp limits of generator g
  — $T_g^{\mathrm{u/d}}$: 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)

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

- Load balance

$$\sum_{g=1}^{G} p_{gt} \geq P_t^{\mathrm{d}} \qquad t = 1, 2, \ldots, T.$$

- Reserve

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

- Power output bounds

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

- Ramp rate bounds

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

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

- Minimum up/downtime

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

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

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

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

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

Appendix 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 column combination primal heuristic is a popular primal heuristic in the column generation procedure in general. The idea of this heuristics is to solve (1) with restricted patterns of solutions, referred to as restricted master IP by Vanderbeck (2005). In the kth iteration, we replace X_s in (1) with $\{x_s^{(k-l)}\}_{l=0,1,2}$ where $\{x_s^{(k-l)}\}_{l=0,1,2}$ are solutions to the sth 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 typically feasible without the need to add artificial variables (columns).

Appendix C: Hyperparameters for the neural networks

The model consists of 4 hidden layers of 1000 units per layer, with skip connections between hidden layers. The tanh function is used as an activation function. This is because in earlier experiments we observed that tanh performed better than ReLU, which is a more standard activation function. As De and Smith (2020) proposed, the weights on the residual connections are initialised to zero. The other weights are initialised based on the methods of Glorot and Bengio (2010). All biases are initialised to zero.

The Adam method (Kingma and Ba (2015)) is used to learn the parameters of the neural network. The learning rate is initialised to 10^{-4}. The model performance is evaluated every 5 minutes, and if it fails to improve for successive 15 minutes, the learning rate is divided by 1.5.

References

Bard JF (1988) Short-term scheduling of thermal-electric generators using Lagrangian relaxation. *Operations Research* 36(5):756–766.

Bertsekas D, Lauer G, Sandell N, Posbergh T (1983) Optimal short-term scheduling of large-scale power systems. *IEEE Transactions on Automatic Control* 28(1):1–11, ISSN 0018-9286.

Borghetti A, Frangioni A, Lacalandra F, Nucci CA (2002) Lagrangian heuristics based on disaggregated bundle methods for hydrothermal unit commitment. *IEEE Power Engineering Review* 22(12):60–60, ISSN 0272-1724.

Briant O, Lemaréchal C, Meurdesoif P, Michel S, Perrot N, Vanderbeck F (2008) Comparison of bundle and classical column generation. *Mathematical programming* 113(2):299–344, ISSN 0025-5610.

De S, Smith S (2020) Batch normalization biases residual blocks towards the identity function in deep networks. Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H, eds., *Advances in Neural Information Processing Systems*, volume 33, 19964–19975 (Curran Associates, Inc.).

Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track* 9:249–256.

Guan X, Luh PB, Yan H, Amalfi JA (1992) An optimization-based method for unit commitment. *International journal of electrical power & energy systems* 14(1):9–17, ISSN 0142-0615.

Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. *The International Conference on Learning Representations*, ULR `https://arxiv.org/abs/1606.01885`.

Nair V, Dvijotham K, Dunning I, Vinyals O (2018) Learning fast optimizers for contextual stochastic integer programs. *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, 591 – 600.

Ostrowski J, Anjos MF, Vannelli A (2012) Tight mixed integer linear programming formulations for the unit commitment problem. *IEEE Transactions on Power Systems* 27(1):39–46, ISSN 0885-8950.

Schulze T, Grothey A, McKinnon K (2017) A stabilised scenario decomposition algorithm applied to stochastic unit commitment problems. *European Journal of Operational Research* 261(1):247–259, ISSN 0377-2217, URL `http://dx.doi.org/10.1016/j.ejor.2017.02.005`.

Takriti S, Birge JR, Long E (1996) A stochastic model for the unit commitment problem. *IEEE Transactions on Power Systems* 11(3):1497–1508, ISSN 0885-8950.

van Ackooij W, Lopez ID, Frangioni A, Lacalandra F, Tahanan M (2018) Large-scale unit commitment under uncertainty: An updated literature survey. *Annals of Operations Research* 271(1):11–85, ISSN 0254-5330.

Vanderbeck F (2005) Implementing mixed integer column generation. *Column Generation*, 331–358 (Springer US), ISBN 0387254854.

Vanderbeck F, Savelsbergh MWP (2006) A generic view of Dantzig–Wolfe decomposition in mixed integer programming. *Operations Research Letters* 34(3):296–306, ISSN 0167-6377.