Evolutionary scheduling of university activities based on consumption forecasts to minimise electricity costs.

1st Julian Ruddick
EVERGi, MOBI
Vrije Universiteit Brussel
Brussels, Belgium
julian.jacques.ruddick@vub.be

2nd Evgenii Genov
EVERGi, MOBI
Vrije Universiteit Brussel
Brussels, Belgium
evgenii.genov@vub.be

***Abstract*—Forecasting and scheduling optimisation are used to reduce the electricity cost of the Monash university. Gradient-boosting method is applied to forecast both generation and consumption time-series. For the consumption forecasts we employ log transformation to model trend and stabilize variance. Additional seasonality and trend features are added to the model inputs when applicable. The forecasts obtained is used as the base load during the scheduling of the activities. The schedule of the activities is obtained through evolutionary optimisation using the covariance matrix adaptation evolution strategy and the genetic algorithm. This schedule is then improved by testing possible times for each activity one-by-one. The battery schedule is formulated as a non-linear programming problem and solved with the Gurobi solver. The results show a good performance for the scheduling of the recurrent activities and the battery. The once-off activity scheduling has still room for improvement.**

I. Introduction

We addressed forecasting and optimization separately. The initial intuition for forecasting the case of the challenge was to use the gradient boosting methods. Large gaps in data and a long forecasting horizon lead to rejecting the idea of using recurrent neural networks and training global models. Furthermore, solutions that employ LightGBM, a popular gradient boosting framework, have dominated in forecasting competitions, particularly the M5 competition [1]. The main challenge with forecasting is identified at preprocessing the data correctly with an outlook for non-stationarities in data. Efforts were also made to tune the model and select the best features. Concerning scheduling optimization we identified two trajectories for finding solution: a heuristic approach and a constraint programming approach. However, the formulation of the scheduling problem has a high level of complexity, which may not be feasible unless broken down into smaller sub-problems. For the activity scheduling, a base schedule is obtained through evolutionary optimisation. This base schedule is then improved by testing possible times for each activity one-by-one. The battery schedule is formulated as a non-linear programming problem and solved with the Gurobi solver [2].

This research was partly funded by VLAIO project MAMUET (grant number HBC.2018.0529).

II. Methods

The approach to forecasting and optimization are illustrated in the data flow diagram in Fig. 1. The forecasting method is highlighted in green and the scheduling is in blue.

A. Forecasting

1) Data pre-processing and feature engineering: The data from the competition was split into training and validation sets. A validation partition is used to tune hyper-parameters, select features and compare the methods' performance. The validation set is selected to start at fixed origin, as the test set is released at once. The last observed values for the month-long period, prior to the forecasted month, is selected to be the validation set for all time series except for Building 5. A significant data drift is observed for that series, so a less recent month set is used for validation.

We calculate the trend and seasonality components for all points of the consumption. The values for these components are also projected into the forecasted period. This is done using the $prophet$ forecasting library [3]. The underlying mode performs a multi-seasonal additive decomposition. Back-testing it showed that using the additional features improves final prediction in all buildings but 5 and 6. Feature selection is handled while tuning the model. Pairwise correlation is calculated using the Pearson correlation method. Features that exceed the correlation coefficient threshold, which is subject to hyperparameter optimization, are discarded. In most cases, feature selection shows to be disadvantageous to the final accuracy on the validation data.

2) Model implementation: We utilize LightGBM [4], a gradient boosting method for solving non-linear regression and classification problems, to forecast the time-series for the month ahead. The objective function is set to minimize mean absolute error (MAE). MAE is proportionate to Mean Absolute Scaled Error (MASE), the metric of the competition. We tune the model for hyperparameters and feature selection on the validation set.

Building 4 time series stands out due to a strictly discrete distribution of consumption values and a high number of

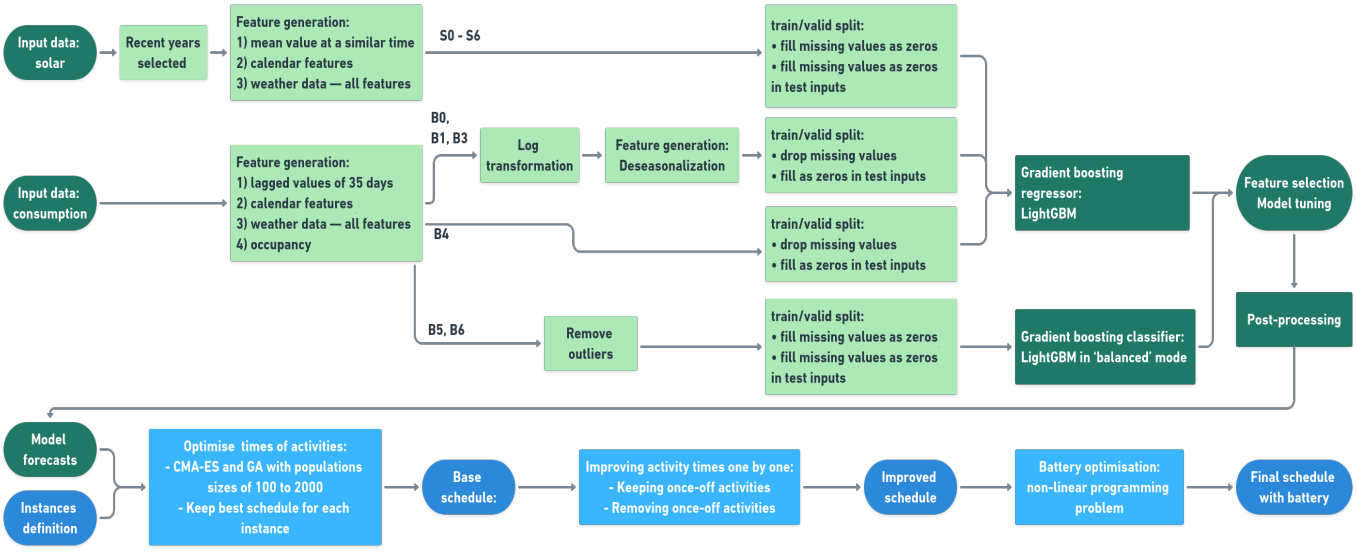


Fig. 1. Data flow of forecasting and schedule optimisation.

missing values. We approach forecasting this series as the multi-class classification problem with an unbalanced dataset. The model is set to 'balanced' mode, where weights are adjusted inversely proportional to class frequencies in the training data

B. Schedule optimisation

Before the start of the optimisation, a *precedence level* is calculated for each activity. The *precedence level* of an activity is the minimum number of days needed before the activity to be able to satisfy the precedence constraints of all the activities. A $level\ after$ value is also calculated for each activity. This value is the minimum number of days needed after the activity to be able to satisfy the precedence constraints of all the activities.

1) Base schedule: The base activity schedule is obtained through the optimisation of an evolutionary algorithm. Two different evolutionary algorithms were tested, the Covariance matrix adaptation evolution strategy (CMA-ES) [5] and the genetic algorithm [6]. The evolution process starts by creating a population of possible schedules. Each individual has two components for each recurrent activity and one component for each once-off activity. The components for the recurrent activities are the day and the time of the day at which the activity should be scheduled. For each activity the days which can be selected depend on how many activity levels are below and above it in the precedence directed graph. The days which can be selected are the five days of the week from which the first x days and the y last days are removed, where x is the $precedence\ level$ of the activity and y is the $levels\ after$ value of the activity. By doing this we discard the days for which it is impossible to schedule the recurrent activity due to the precedence constraints. The time of the day for the recurrent activities is selected from the start of the working day to the end of the working day minus the duration of the activity. The time for the once-off activities is selected from the first time of the month to the last time of the month minus the duration of the activity (Alg. 1 line 2).

To satisfy the precedence constraints for the recurrent and once-off activities, the days of the activities are changed gradually starting from the activities with the lowest precedence level. If an activity has a precedence scheduled for the same or a later day, the activity will be rescheduled to the day after the latest day of its precedence activities. The time of the day stays the same for the rescheduled recurrent and once-off activities. If due to this process a once-off activity is scheduled after the last day of the month, this activity is discarded (Alg. 1 line 4).

When the precedence constraint has been satisfied, the room constraints need to be enforced. Rooms are assigned first for the recurrent activities and second for the once-off activities using the same process. Activities for which the product of the duration and the number of rooms occupied by the activity is the highest are the first to be assigned to a room. An activity is assigned to rooms available with the lowest ids first. If there are not sufficient rooms available to fit the activity, the activity is scheduled at a different time of the day and during working hours for the recurrent activities. This process tests times gradually from closest to furthest to original given time. If no rooms are available in the selected day, the solution is discarded by giving it a high score of 200 000 (Alg. 1 line 5).

Once-off activities which increase the score of the objective function from the previously obtained feasible schedule are removed. For each once-off activity, the electricity cost of running the activity, the value and the penalty of itself and all the activities necessary to schedule this activity are summed to give the benefit of scheduling the activity. This benefit defines the impact of the once-off activities has on the objective function without taking the electricity consumption peak cost

into account. The more negative this benefit is, the more it will decrease the objective function. The activity with the largest negative benefit and all the activities necessary to schedule this activity are kept for the final solution. The same process is repeatedly executed with the electricity cost, the activity value and the penalty of the kept activities set to 0 until the reward of all remaining activities are positive, in which case the remaining activities are discarded. This process removes all once-off activities, which increase the objective function score. Some kept once-off activities may still increase the objective function score via the peak cost (Alg. 1 line 2).

The obtained schedule is evaluated through the objective function and the score is given to the evolutionary algorithm (Alg. 1 line 7). Moreover, the evolutionary algorithm optimises the selected days and times of the recurrent and once-off activities to minimise the objective function scores of the obtained schedules (Alg. 1 lines 8 and 2).

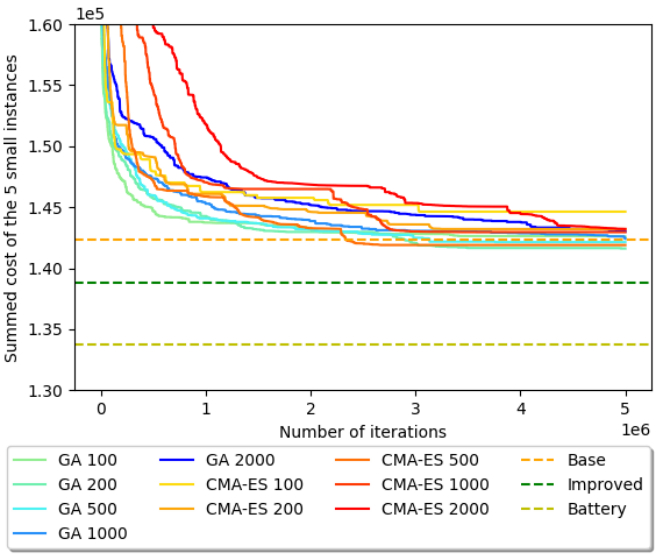
The CMA-ES and GA were tested with population sizes from 100 to 2000 as is shown in Fig. 2. The stopping criteria for the CMA-ES is met when the f tolerance is smaller than 100 or the x tolerance is smaller than 1. The stopping criteria for the GA is met when no improvement larger than 1 is found for 500 generations in a row. Both stopping criteria were tuned to stop the evolution process when no or very slight improvements were found for multiple generations. The starting σ for the CMA-ES is 0.5. The GA selects 10% of each population as parents. All other parameters of the GA are the default parameters from the PyGAD library [7].

For small instances CMA-ES and the GA both get close or exceed with all populations size the base schedule used for the competition submission (see Fig. 2). However for the large instances, CMA-ES outperforms the GA for all population sizes except 2000. The base schedule used for the competition submission had 3 small schedules obtained with the GA and the 7 others obtained with CMA-ES.

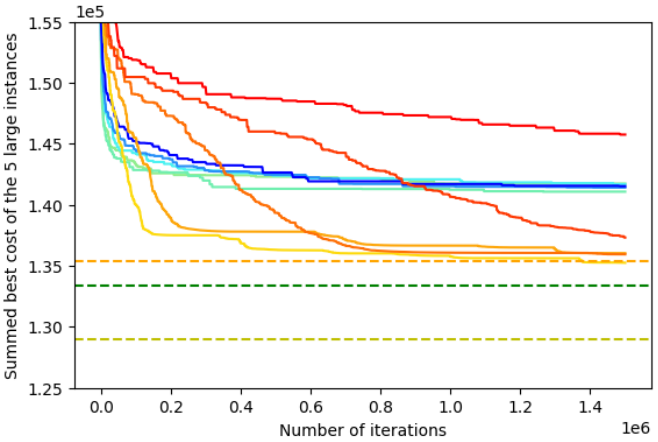
Algorithm 1 Obtain base schedule

Input: instance
Output: base schedule
1: **while** stop criteria is false **do**
2: Evolutionary algorithm generate population within possible times
3: **for** individual in new population **do**
4: Enforce precedence constraints and define day and time of all activities
5: Define rooms for all activities
6: Remove once-off activities with negative impact on electricity cost
7: Calculate electricity cost
8: Evolve population based on previous individuals and associated electricity costs
9: **return** base schedule

2) Improved activity schedule: From the base schedule, an improved schedule is obtained by modifying the time of the activities one by one. This improvement is done in two versions, one improving the base schedule and a second improving the base schedule from which the once-off activities are removed (Alg. 2 line 2). The improvement starts by modifying the times of the recurrent activities then the once-off activities (Alg. 2



(a) Small instances



(b) Large instances

Fig. 2. Visualisation of the optimisation process to find a base schedule with the GA and the CMA-ES approach for different population sizes. Each moving line represents the summed best cost obtained for the 5 instances of each category (small in Fig. 2a and large in Fig. 2b) during consecutive optimisation processes. Once the evolutionary process reached its stopping criteria, a new evolutionary process for the same instance is started. The horizontal lines are the scores obtained by the schedules used for the competition submission.

line 4). The time of the activities are modified one by one keeping the time of the other activities fixed and starting with the activities with the lowest *precedence level* (Alg. 2 lines 5 and 6). Once a better time for an activity is found, the better time is given to that activity and the process continues with this new improved solution. In a first phase, the activities try all times within the days that allow all activities to be scheduled and that respect the precedence constraints of the activities already in the schedule (Alg. 2 line 8). In a second phase, the activities try all times that respect the precedence constraints of the activities already in the schedule (Alg. 2 line 10). This means that in the second phase the once-off activities can try times that discard other once-off activities to be scheduled

due to precedence constraints. Overall this process finds better times for the once-off and recurrent activities and schedules new once-off activities that were previously not scheduled.

All improvement methods for the activity schedule reduce the final cost (see Fig. 3). The methods where the once-off activities are removed seems slightly better on average but the method where the once-off activities are kept does yield in some cases better results. The improved schedule used for the competition submission had 4 schedules obtained by keeping the once-off activities and 6 obtained by removing the once-off activities.

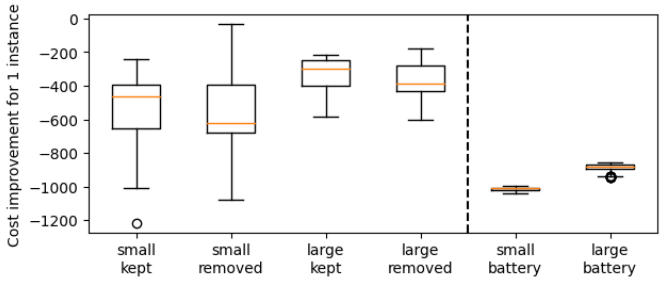


Fig. 3. Box-plots of the different methods used to improve the base schedule and the battery schedule. The box-plots for the improvement methods are generated on 40 values each, which are the costs of the improvement of the 8 best base solution found for each instance. The box plots for the battery schedule contain 80 values each, which are the cost obtained by adding the battery schedule to the 160 improved schedules in the other box-plots.

Algorithm 2 Improve base schedule

Input: base schedule
Output: improved schedule
if remove once off = true **then**
  remove once off activities from base schedule
for $i := 1$ to 2 **do**
  for *activity* in all activities starting with recurrent **do**
    for *day* in days of the month **do**
      if *activity* precedence level = *day* **then**
        if $i = 1$ **then**
          improve time of *activity* within recommended days
        else if $i = 2$ **then**
          improve time of *activity* within all times
return improved schedule

3) Battery schedule: The schedule of the batteries is found with the activity times of the improved schedule. The behaviour of a battery and the objective function are modelled as a non-linear mathematical programming problem using the constraints and objective function described in the problem description. The Gurobi solver [2] was used to minimise the objective function with the variables of the problem being the decision to charge, hold or discharge the batteries and this action modifying the final load of the improved solution. This process calculates the perfect schedule of the battery for the given improved schedule and the imperfect forecast of the buildings and solar panels loads. Fig. 3 shows that the battery schedule decreases the final score more than the schedule improvement. For both the small and large instances, all the battery improvements are at maximum within a cost value of 100 from each other. This tends to indicate that the improvement from the battery does not depend much on the activity schedule so doing the battery scheduling separately from the activity scheduling is reasonable.

C. Example schedule and forecast

Fig. 4 shows the best activity schedule found for the small_0 instance. The recurrent activities seem well scheduled to keep a low max power value and to avoid high electricity prices. The once-off activities are placed mainly during working hours and multiple are scheduled in the last two days which have low electricity price and no recurrent activity. There is nonetheless a visible improvement that can be made to this schedule, 7 out of the 20 once-off activities have not been scheduled and between time-steps 2676 and 2709 the prices are negative. Shifting once-off activities from the 2nd last day to this time-slot and adding non-scheduled once-off activities to the last two working days would certainly decrease the final cost. A better optimisation process should solve this issue.

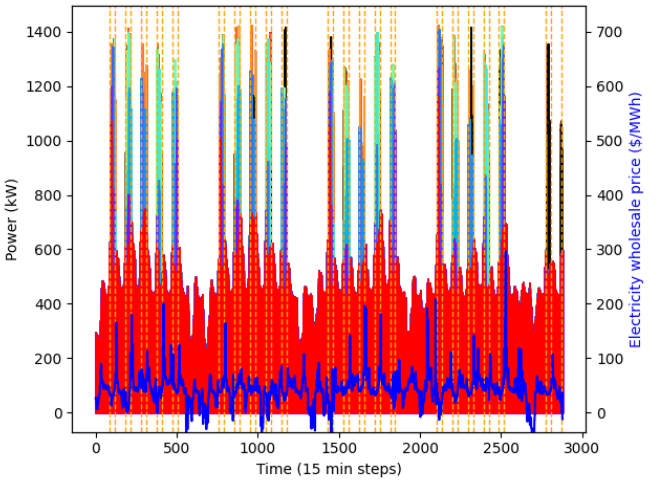


Fig. 4. Representation of the best found improved schedule for the small_0 instance. In red is the forecasted load of the buildings minus the production of the solar panels. The load from the recurrent activities are represented in different colors and the load from the once-off activities are represented in black. The orange vertical lines represent the start and the end of the working hours of the problem. The blue line is the price of price of electricity.

References

[1] S Makridakis, E Spiliotis, and V Assimakopoulos. The m5 accuracy competition: Results, findings and conclusions. *Int J Forecast*, 2020.

[2] LLC Gurobi Optimization. Gurobi Optimizer Reference Manual, 2021.

[3] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[4] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[5] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, March 2003. Conference Name: Evolutionary Computation.

[6] David E Goldberg. *Genetic Algorithms*. Pearson Education India, 1st edition edition, 2006.

[7] Ahmed Fawzy Gad. PyGAD: An Intuitive Genetic Algorithm Python Library. *arXiv:2106.06158 [cs, math]*, June 2021. arXiv: 2106.06158.