A local search method for solving a bi-level timetabling and battery scheduling problem

1st Qingling Zhu, 2th Yaojian Xu, 3nd Minhui Dong, 5th Junchuang Cai, 6th Junkai Ji, 6th Qiuzhen Lin
College of Computer Science and Software Engineering,
Shenzhen University,
Shenzhen, China
{zhuqingling,xuyaojian2021,dongminhui2019,caijunchuang2020}@email.szu.edu.cn,{jijunkai,qiuzh.lin}@szu.edu.cn

4rd Wu Lin, 7th Kay Chen Tan
Department of Computing,
The Hong Kong Polytechnic University,
Hong Kong, SAR China
21039144r@connect.polyu.edu.hk, kaychen.tan@polyu.edu.hk

***Abstract*—In this paper, we will present a summary of the methods used by the authors to solve the IEEE-CIS technical challenge on Predict+Optimize for renewable energy scheduling. We will treat this challenge problem as a bi-level optimization problem. In the top level, a timetabling problem is solved by local search method with relaxation strategy. In the lower level, battery scheduling problem is solved after applying the optimized timetabling.**

***Index Terms*—battery schedule, lecture schedule, optimization, prediction**

I. Introduction

This summary paper describes the method used to solve the prediction and optimization challenge problem of the IEEE-CIS competition. Overall, we were able to solve 5 small scale and 5 large scale problem instances.

II. Methodology

A. Timetabling

a) Preprocessing for activities: Activities cannot arbitrary scheduled since they have precedences. After preprocessing, we can get the feasible timeslots for each activity. Table I shows the feasible days can be scheduled for each recurring activity. As can be observed from this table, there are 9 activities that can only scheduled at Monday, 5 activities that can only scheduled at Tuesday, 2 activities that can only scheduled at Wednesday, 2 activities that can only scheduled at Thursday, 2 activities that can only scheduled at Friday. The same preprocessing can be done for once-off activities.

b) Solution representation for timetabling: First, we describe the algorithm used to find feasible solutions. Basically, this is a stochastic algorithm, assigning classes at random. After preprocessing, we obtain the feasible period for each activity as shown in the last column of Table I. To formulate a timetable, we need to determine the following two aspect.

- The order of activities with same scope on the same day. For example, activities r0, r3, r7, r14, r19, r26, r27, r35, and r43 can be scheduled on Monday and they all have scope 1. Therefore, the order of these 9 activities is a permutation of $\{1$ to $9\}$. For activities r4 and r8, they can be scheduled on Monday and Tuesday. The order of these activities is a permutation of $\{10,11\}$.
- Timeslot selection: Since each activities has its own feasible timeslots to select (the last column in Table I), timeslot selection should be determined in a solution representation.

Base on these two aspects, we can represent a feasible timetable by a vector with length $2n$ where n is the number of activities. The pseudo-code of local search method for timetabling is illustrated in Algorithm 2.

TABLE I
Information after preprocessing for instance-small-0

begin	activity	n	scope	period
Monday	r0[1,1],r3[1,1],r7[1,1],r14[1,1],r19[1,1], r26[1,1],r27[1,1],r35[1,1],r43[1,1]	9	1	P(0,31)
	r4[1,2],r8[1,2]	2	2	P(0,63)
	r40[1,3]	1	3	P(0,95)
	r33[1,4]	1	4	P(0,127)
	r12[1,5],r18[1,5],r37[1,5],r47[1,5]	4	5	P(0,159)
Tuesday	r1[2,2],r13[2,2],r22[2,2],r46[2,2],r49[2,2]	5	1	P(32,63)
	r9[2,3],r17[2,3],r21[2,3],r24[2,3]	4	2	P(32,95)
	r29[2,4],r32[2,4],r34[2,4]	3	3	P(32,127)
	r16[2,5]	1	4	P(32,159)
Wednesday	r20[3,3],r25[3,3]	2	1	P(64,95)
	r5[3,4],r39[3,4],r44[3,4]	3	2	P(64,127)
	r6[3,5],r10[3,5],r11[3,5],r15[3,5], r28[3,5],r30[3,5],r36[3,5],r48[3,5]	8	3	P(64,159)
Thursday	r41[4,4],r45[4,4]	2	1	P(96,127)
	r2[4,5],r38[4,5],r42[4,5]	3	2	P(96,159)
Friday	r23[5,5],r31[5,5]	2	1	P(128,159)

B. Battery scheduling

The solution representation for battery scheduling is very simple. We use three state, i.e., hold, discharge, charge, for each timeslot and each battery. Then a local search method is

Algorithm 1: Evaluation

Input: permutation
battery schedule
Output: solution and its score

```
/* recurring activity scheduling */
for each recurrent activity do
    for each feasible time slot do
        check whether it is feasible to schedule current
          activity to current timeslot
        if feasible then
            calculate cost when this activity sheduled at
              current timeslot
            add this cost to allCost
    sort allCost according to cost
    select a timeslot from the best 10 timeslots
/* once-off activity scheduling */
for each once-off activity do
    for each feasible time slot do
        check whether it is feasible to schedule current
          activity to current timeslot
        if feasible then
            calculate cost when this activity sheduled at
              current timeslot
            add this cost to allCost
    sort allCost according to cost
    select a timeslot from the best 10 timeslots
```

Algorithm 2: TimeTabling Algorithm

Input: phase2-instance-small,large-0-5.txt
Output: time-tabling

```
Baseload = total build load - total solar production
function PreProcess
    Get OrderedRCourse/* sort Recurring courses
       according to their range of feasible
       timeslots. */
    Get OrderedACourse/* sort once-off activity
       according to their range of feasible
       timeslots. */
random generate a combination/* refer to the
   solution representation in II-A0b */
cost = evaluate this combination/* Algorithm 1 */
while stop criterion is not met do
    disturb current combination
    evaluation current combination
    if cost < bestCost then
        bestCost=cost
        save this combination
```

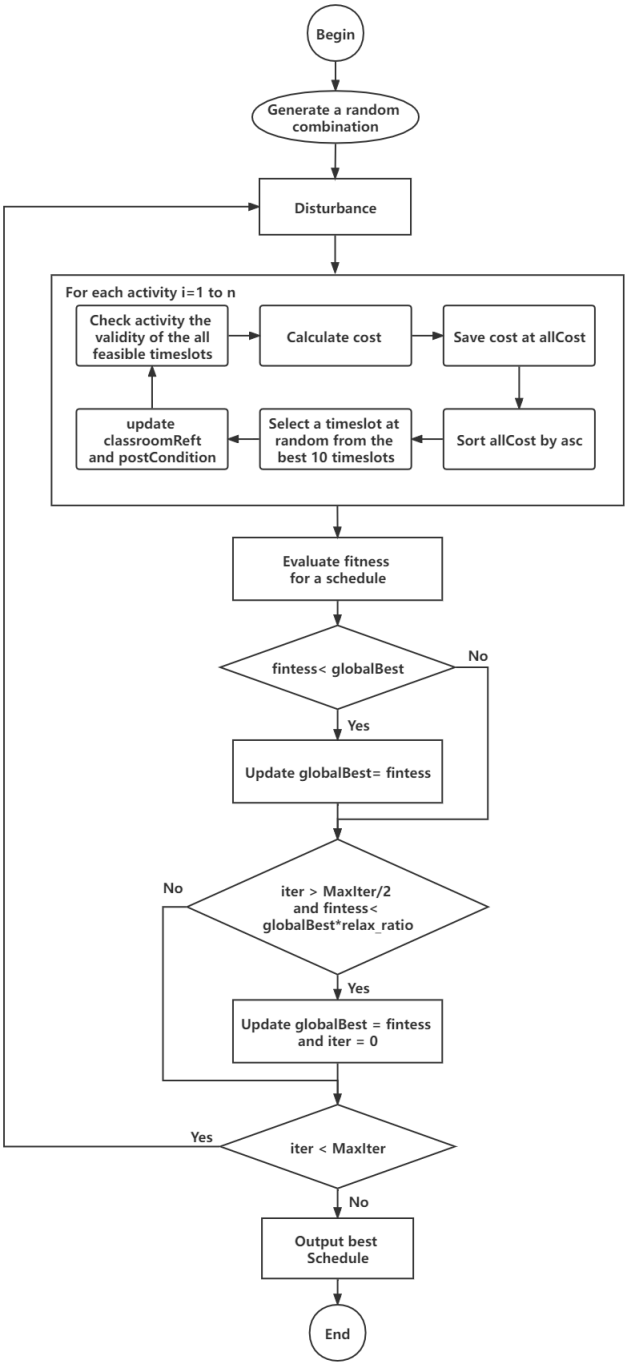

Fig. 1. The flowchart of timetabling problem

Algorithm 3: Battery Schedule Algorithm

Input: p_m: permutation rate, set to $\frac{1}{\#\text{timeslots}}$

read load information after timetabling and calculate maxload
random generate a feasible solution and calculate its score
set $iter = 0$
while $iter < maxIter$ **do**
  for *each bit of the current solution* **do**
    if $rand < p_m$ **then**
      random select a state for current bit from {hold,discharge,charge}/* make sure the solution is feasible */
  evaluation current battery schedule
  if $cost < bestCost$ **then**
    bestCost=cost
    save this battery schedule
return bestSolution

Algorithm 4: Complete bi-level optimization procedure

Input: instance-small,large-0-5.txt
Output: complete solution

read instance information
get baseload and calculate maxload
initialize battery schedule as all zeros/* hold */
while *true* **do**
  /* fix battery to optimize timetabling */
  Timetabling /* **Algorithm 2** */
  if *timetabling cannot get a better solution* **then**
    disturb the current timetabling/* relaxation */
  /* fix timetabling to optimization battery */
  battery schedule algorithm /* **Algorithm 3** */
  if *!isImproved* **then**
    break
return the current best solution

used to search the best battery scheduling. The pseudo-code of battery scheduling procedure is illustrated in Algorithm 3.

The whole bi-level optimization procedure is illustrated in Algorithm 4.

C. Forecast

In this subsection, we will discus how forecast and optimisation interact.

a) Feature engineering: In this competition, we are provided with historical hourly weather data but need to make predictions on 15-minute-level, so the original weather features and the target cannot achieve one-to-one correspondence. We take hour, minute, and weekday as new features to do 15-minute-level prediction. In addition, the "total demand" of Victoria that provided for the optimization problem, was find has a strong correlation with the energy demand of 6 buildings, thus this feature was used in the energy demand prediction.

1) The processing of energy demand of 6 buildings: According to the characteristics of the data set of each building, we have made a corresponding preprocess for it.

a) Building 0: Building 0's four demands in one hour are always the same, so we only need to predict a quarter of its data, and we use 0 minutes per hour for training and prediction. For example, if the predicted value of 1:00 on November 2 is 100, then the values of 0:15, 0:30 and 0:45 on November 2 are 100, too.

b) Building 1: We found that there were regularities among the data of 0, 15, 30 and 45 minutes per hour in building 1. Therefore, we divided them into four groups depending on the minute for prediction. In addition, we found that there were significant differences between the data of Building 1 on weekdays and weekends. Therefore, on the basis of the previous processing, we split the data into weekdays and weekends to make predictions respectively.

c) Building 3: Building 3's four demands in one hour are always the same, so we only need to predict a quarter of its data. The data of Building 3 on weekdays and weekends have significant differences, so we split the data into weekdays and weekends to make predictions respectively.

d) Building 4: Building 4 has a large number of missing values, and we did not finally explore an effective scheme to predict its value. However, when we counted the months that has more than 50% effective values, we found that the data with value 1 has a huge proportion in all data, so we finally set all of them to 1.

e) Building 5: Building 5 also has a large number of missing values. After trying various methods, we ended up doing no extra processing on it.

f) Building 6: The data of Building 6 on weekdays and weekends have significant differences. After trying various methods, we ended up using the same method as building 1.

2) The processing of power production of 6 solar panels: We find that solar energy generation is highly correlated with the "surface_solar_radiation". When "surface_solar_radiation" is 0, more than 99% energy generation of solar 0 is 0.01, while more than 99% energy generation of other power panels is 0. Therefore, in the predicted value, we conducted corresponding data post-processing according to the value of "surface_solar_radiation" in November.

3) Models: In this prediction, several machine learning models were used, including LSTM, LSTM, GRU, SVR, ELMAN, MLP, DNR, Prophet. For each set of data, we divide it into three parts. Data for the whole month of November was used as the test set. Part of the data sets at the end of October was used as the validation set. The remaining data of October and the data of August and September were used as the training set. We integrate several models with high

performance in the validation set and take their average value as the final result.

4) Discussion: we think the forecast accuracy is not directly relevant to the optimization results. The reasons are two folds.

1) missing values: the missing values are removed when evaluation for prediction. However, they are treated as zeros when optimization.
2) the accuracy of baseload, which equals the overall building load minus the overall solar production, is directly relevant to the optimization results. Therefore, we can use two models, one model to predict the overall building load and one model to predict the overall solar production. We think this can reduce the overall error compared to 12 models (6 models for building load and 6 models for solar production).

III. Experiments

In the experimental part, we use the above introduced algorithm to search for the best timeslot for a set of activities and the best timeslot for battery charge and discharge. In the beginning, an activity schedule that meets the constraints is randomly generated. After several round of iterations, an optimal activity schedule is found and saved. When optimizing battery, the optimal activity schedule is fixed, and the charging and discharging decision of each timeslot for each battery is determined. Finally, an optimal solution is generated.

In Fig. 2 and Fig. 3, the predicted baseload, load after timetabling, load after battery schedule for all the 10 adopted test instances are plotted for illustration. Notice that the Y-axis on the right is price. There are 2880 timeslots were loaded from 11-1 11:00 to 12-1 10:45. Based on the information shown in these figures, it can be seen that on the premise of accurate prediction, the algorithm we use shows very good advantages in solving the combinatorial optimization problem of activity arrangement. Next, I will describe the observation from these figures in detail, using the first image as an example.

1) The overlap of battery-baseload and TT-baseload in the figure indicates that timeslot is not charged or discharged by batteries, for example, from 0:00 to 9:00 on November 11-2. The overlap between baseload and TT-baseload indicates that the timeslot has no activities scheduled, such as the two-day weekend of 11-7th to 11-8th. Where Baseload is not covered, baseload is almost covered by TT-baseload in other periods. This shows that our algorithm schedules one-off activities as much as possible in working days under the condition that repetitive activity constraints are satisfied, to get rewards.
2) Battery-baseload has four consecutive peaks, representing four consecutive weeks in November. The load from Monday to Friday is much higher than that on weekends, indicating that one-off activities are arranged during working hours.
3) In the cost function, the highest peak will bring an additional penalty, so the battery must discharge at the moment of the highest peak to reduce the peak value. Therefore, in the iteration algorithm of the battery design stage, the general idea of battery charge and discharge is as follows: discharge at high load, charge at low load and charge at low electricity price under the condition of satisfying battery constraints. The highest peak value of TT-baseload roughly occurred during the working hours of 11-12 and 11-13. It can be seen from the figure that battery-baseload decreased significantly in these two stages. This is enough to show that our algorithm in the battery optimization stage plays a very good effect. It should be noted that the battery optimization phase is an iterative process to reduce the load. As shown in the figure, the peak value of battery-baseload is stable for four consecutive weekdays, while on non-weekdays battery-baseload is slightly higher than TT-baseload, which is exactly what we want to see.
4) There was a slight difference between the working hours of Monday, November 9th and Monday, November 16th, due to the arrangement of one-off activities and the charging and discharging of batteries.
5) Battery-baseload must be lower than TT-baseload at the beginning of 11-1 11:00 because the battery is fully charged and must be discharged first. The last two timeslots are always charged for all 10 instances because the prices are below zero.

A. Figures and Tables

The optimization results under predicted base load is shown in Table II. The total score under the predicted base load is 264028, and the total score under the "true" base load is 342810. The gap between these two results is 78782. This is a very large gap and should be reduced using other strategies. One strategy is improving the accuracy of baseload. I don't know how the true baseload, which is used to evaluation our results in server, is got for 2020-11 because the students are enrolled to the university.

TABLE II
Results under predicted base load

instance	score
phase2_instance_small_0	27702
phase2_instance_small_1	26070
phase2_instance_small_2	26236
phase2_instance_small_3	28237
phase2_instance_small_4	26222
phase2_instance_large_0	25836
phase2_instance_large_1	26399
phase2_instance_large_2	25227
phase2_instance_large_3	26074
phase2_instance_large_4	26025
total score (predicted)	264028
total score (true)	342810

Fig. 2. The plots of baseload, load after timetabling, load after battery schedule and price during the whole 2880 timeslots for small type instances

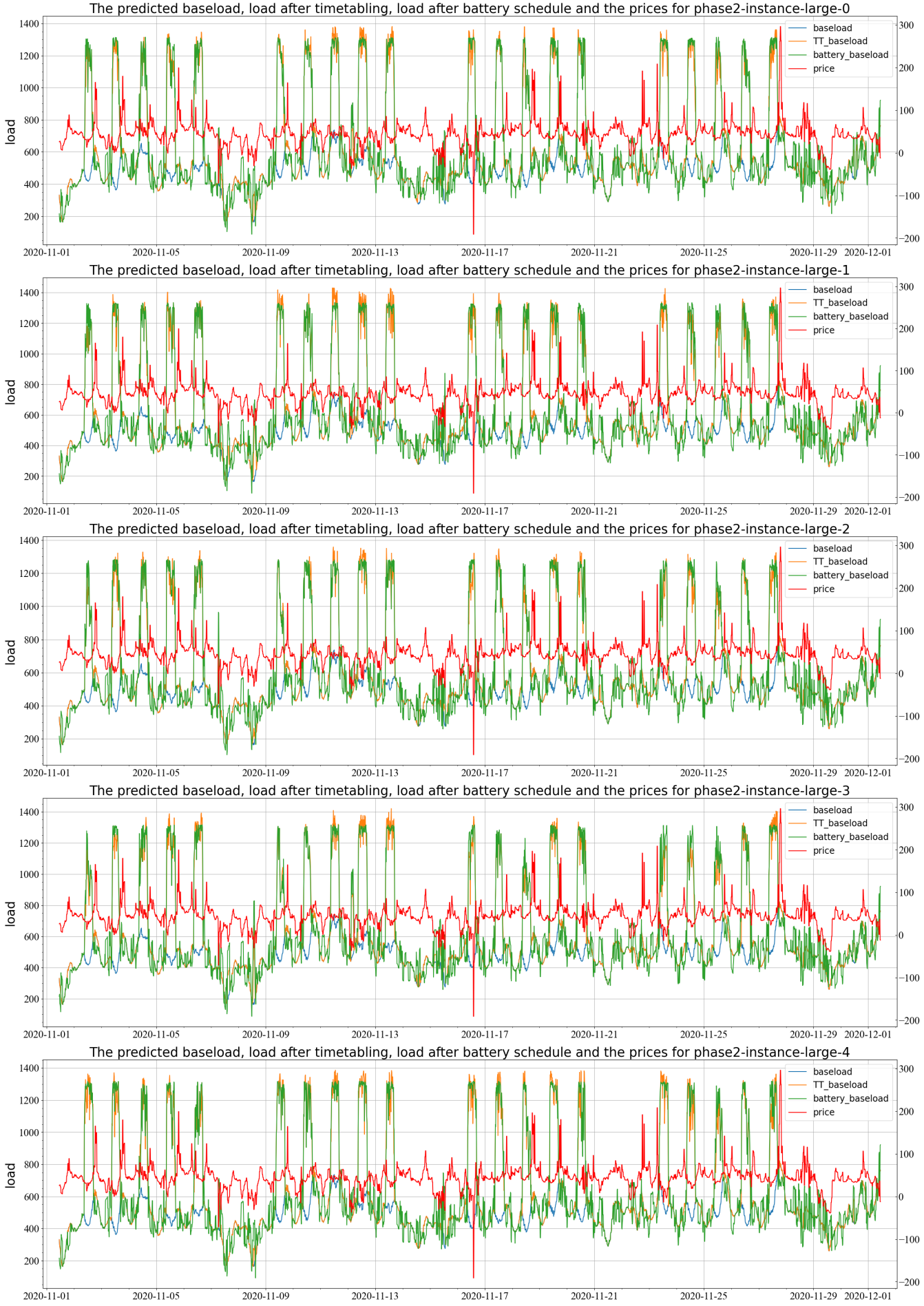

Fig. 3. The plots of baseload, load after timetabling, load after battery schedule and price during the whole 2880 timeslots for large type instances