Project Report: Optimizing demand-side strategies

II. Code submission and result reproducibility

Please see the folder simulate.zip attached in the email where you can find two files:

- battery_controller.py: our code
- readme.txt: show necessary tools and the explanation of our linear programming model

III. Model documentation and write-up

1. Who are you (mini-bio) and what do you do professionally?

Minh Hoàng Hà: (*ORLab* on the forum)

Mini-bio: Minh Hoàng Hà is currently a lecturer at University of Engineering and Technology, Vietnam National University. His research interests include combinatorial optimization problems such as packing, scheduling and routing.

Duy Mạnh Vũ: (*duymanhit* on the forum)

Mini-bio: Duy Mạnh Vũ is currently a second-year undergraduate student at University of Engineering and Technology, Vietnam National University. He likes studying operations research and machine learning.

Thị Thanh Lâm Vũ
Mini-bio: Thị Thanh Lâm Vũ is studying Computer Science at University of Engineering and Technology, Vietnam National University as a second-year undergraduate student. Her main interests are optimization methods applied in transportation and electric production.

Văn Phú Hoàng
Mini-bio: Văn Phú Hoàng is a second-year undergraduate student at University of Engineering and Technology, Vietnam National University. His plan in the near future is working on data mining and optimization.

Tiến Thành Đàm
Mini-bio: Tiến Thành Đàm is an undergraduate student at University of Engineering and Technology, Vietnam National University. He decided to follow optimization and data mining after working on the project "Power Laws: Optimizing Demand-side Strategies" of ORLab organized by DrivenData.

2. High level summary of your approach: what did you do and why?

We considered the problem as a dynamic optimization problem. The problem at each step was modeled as a linear programming (LP). We selected Ortools to solve LP model optimally because it seemed to be the fastest and easy to install on the docker.

The next idea comes from the fact that in some cases where the coming PVs are enough to fully charge the battery, the solution provided by Ortools suggests us to charge a quantity of energy that is higher than the real difference of production and consummation. And as a result, we have to buy electric from the grid to supplement the variance in the forecast. We then tried to adjust the solution obtained from Ortools to take into account the uncertainty of the forecast such that the value of the new solution objective function is kept the same. To do so, from the obtained solution, we will find a period *p* (normally in daytime) in which the battery is consecutively not discharged and ($pv\_i - load\_i \geq 0$) for each step *i* in *p*. We then try to charge a quantity of ($pv\_i - load\_i - reducedEnergy$) for each step during the period. Here, *pv i* and *load i* are the forecasting amount of production of PV and consummation of building at step *i* respectively; *reducedEnergy* is a fixed amount of energy that is reduced equally in all steps of the period *p*. We will find the maximal value of *reducedEnergy* so that the total quantity of charged energy in the considering period is kept the same as in the optimal solution provided by Ortools and the power limit constraint is still satisfied. Note that, the larger the value of *reducedEnergy* is, the more the risk of buying energy from grid decreases. The binary search is used to compute *reducedEnergy*. The similar idea is applied for the period where the battery is consecutively not charged and ($pv\_i - load\_i \leq 0$) for each step *i* in the period.

3. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

   First part:

```
# Ortools
solver = pywraplp.Solver("B",
pywraplp.Solver.GLOP_LINEAR_PROGRAMMING)

#Variables: all are continous
charge = [solver.NumVar(0.0, limit, "c"+str(i)) for i in range(number_step)]
dis_charge = [solver.NumVar( dis_limit, 0.0, "d"+str(i)) for i in
range(number_step)]
battery_power = [solver.NumVar(0.0, capacity, "b"+str(i)) for i in
range(number_step+1)]
grid = [solver.NumVar(0.0, solver.infinity(), "g"+str(i)) for i in
range(number_step)]

#Objective function
objective = solver.Objective()
for i in range(number_step):
   objective.SetCoefficient(grid[i], price_buy[i] - price_sell[i])
   objective.SetCoefficient(charge[i], price_sell[i] + price_buy[i] / 1000.)
   objective.SetCoefficient(dis_charge[i], price_sell[i])
objective.SetMinimization()

# 3 Constraints
c_grid = [None] * number_step
```

```
c_power = [None] * (number_step+1)

# first constraint
c_power[0] = solver.Constraint(current, current)
c_power[0].SetCoefficient(battery_power[0], 1)

for i in range(0, number_step):
   # second constraint
   c_grid[i] = solver.Constraint(energy[i], solver.infinity())
   c_grid[i].SetCoefficient(grid[i], 1)
   c_grid[i].SetCoefficient(charge[i], -1)
   c_grid[i].SetCoefficient(dis_charge[i], -1)
   # third constraint
   c_power[i+1] = solver.Constraint( 0, 0)
   c_power[i+1].SetCoefficient(charge[i], charging_efficiency)
   c_power[i+1].SetCoefficient(dis_charge[i], discharging_efficiency)
   c_power[i+1].SetCoefficient(battery_power[i], 1)
   c_power[i+1].SetCoefficient(battery_power[i+1], -1)

#solve the model
solver.Solve()
```

This piece of code is the backbone of our approach. We formulate the problem as a linear programing model and solve it by Ortools. At the end of this step, we have the optimal way to control the battery charging system with respect to forecast data for next 96 steps.

Second part

```
If ((energy[0] < 0) & ( dis_charge[0].solution_value() >= 0)):
        n = 0
        first = -limit
        mid = 0

        sum_charge = charge[0].solution_value()
        last = energy[0]
        for n in range(1, number_step):
           if ((energy[n] > 0) | (dis_charge[n].solution_value() < 0) | (price_sell[n]
!= price_sell[n-1])):
              break
           last = min(last, energy[n])
           sum_charge += charge[n].solution_value()
        if (sum_charge <= 0.):
            return battery_power[1].solution_value() / capacity
        def tinh(X):
           res = 0
```

```
            for i in range(n):
                res += min(limit, max(-X - energy[i], 0.))
            if( res >= sum_charge): return True
            return False
        last = 2 - last
        # binary search
        while (last - first > 1):
            mid = (first + last) / 2
            if (tinh(mid)): first = mid
            else: last = mid
        return (current + min(limit, max(-first - energy[0] , 0)) *
charging_efficiency) / capacity
```

This code is for the purpose of reducing the negative impact of the imperfection of consumption and production predictions if in the solution provided by Ortools, the battery is not discharged in a number of consecutive steps. Here we use the *binary search* to compute *first* (or *reducedEnergy* mentioned aboved), the maximal quantity of energy that we should subtract at each step to reduce the risk of buying unnecessary energy from the grid.

Third part

```
If ((energy[0] > 0) & (charge[0].solution_value() <=0)):
        n = 0
        first = dis_limit
        mid = 0
        sum_discharge = dis_charge[0].solution_value()
        last = energy[0]
        for n in range(1, number_step):
            if ((energy[n] < 0) | (charge[n].solution_value() > 0) | (price_sell[n] !=
price_sell[n-1]) | (price_buy[n] != price_buy[n-1])):
                break
            last = max(last, energy[n])
            sum_discharge += dis_charge[n].solution_value()
        if (sum_discharge >= 0.):
            return battery_power[1].solution_value() / capacity

        def tinh2(X):
            res = 0
            for i in range(n):
                res += max(dis_limit, min(X - energy[i], 0))
            if (res <= sum_discharge): return True
            return False
        last += 2

        # binary search
```

```
            while (last - first > 1):
                mid = (first + last) / 2
                if (tinh2(mid)): first = mid
                else: last = mid
            return (current +  max(dis_limit, min(first - energy[0], 0)) *
discharging_efficiency) / capacity
        return battery_power[1].solution_value() / capacity
```

This code has the same idea as the one above. But now the battery is not charged in a number of consecutive steps.

4. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

   We have tried to use several open-source linear programming tools such as pulp, scip, glpk. Only pulp could be installed on the docker. However, pulp is much slower than Ortools.

5. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

   No, we didn't.

6. How did you evaluate performance of the model other than the provided metric, if at all?

   Other than the provided metrics, we mainly evaluated the performance of our model on the aspect of running time. In the beginning, we were afraid that solving a liner programming model for 96 steps would take a lot of time while we had a large number of simulations to run. And we had to formulate the problem carefully so that its size should be as small as possible.

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

   No, I think our model can run normally on common platforms without any problem.

8. Do you have any useful charts, graphs, or visualizations from the process?

   No, we don't.

9. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

In the future, we could improve the current result by proposing a new linear programing model that takes into account the uncertainty of the forecast. This model should use new parameters that reflect the variance of the forecast of pv and load at each step and can be easily computed from the past data.