COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Columbia Peak Shaving

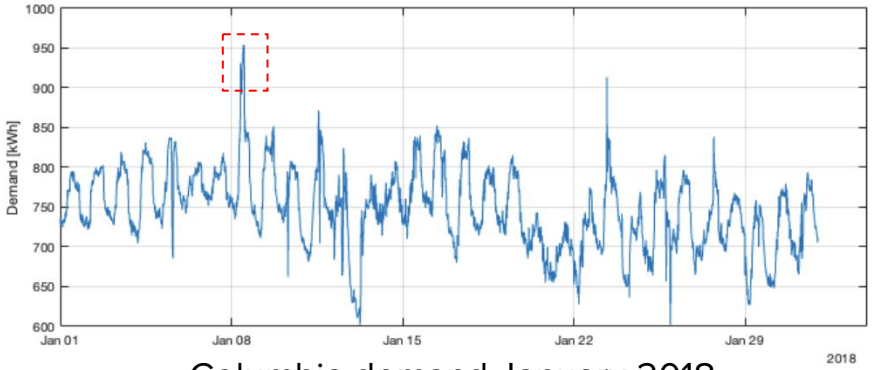
Gabrielle Nyirjesy
Rafael Auada
Steven Yang

Columbia's two part electricity bill

Energy consumption

Peak demand

Demand-supply contract: $ 0.13 / kWh



Columbia demand January 2018

On-peak / Off-peak tariffs

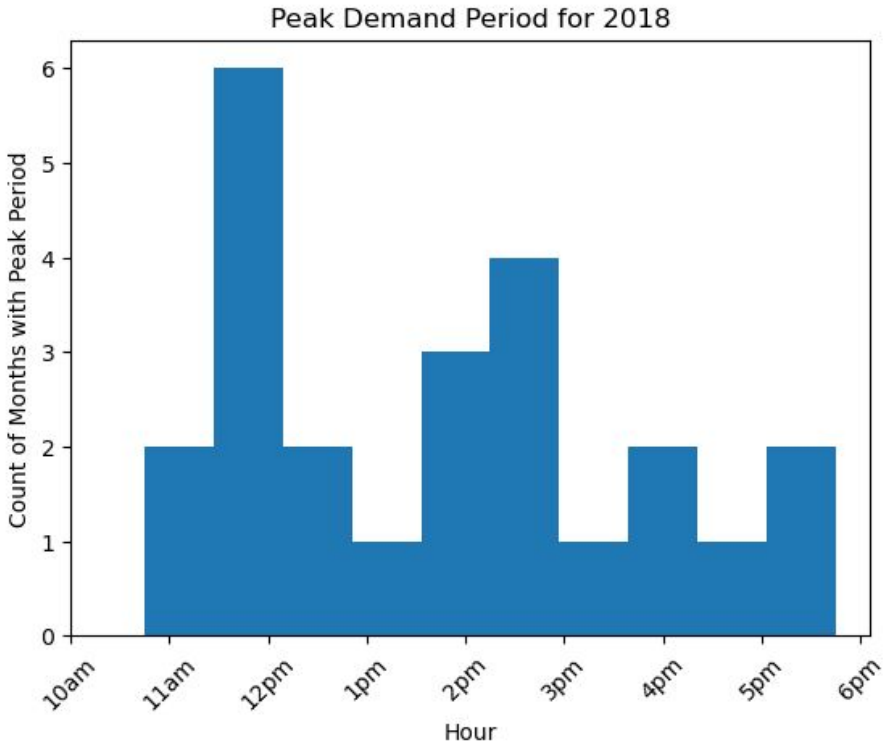
Objectives

- What technology: lithium-ion battery or thermal energy storage?
- How should storage be operated?
- Total cost savings that can be expected.

Battery Technology	Lithium Ion	Thermal
Power rating ($/kW)	300	500
Energy rating ($/kWh)	200	50
Energy duration (hours)	4	12
Single-trip efficiency (%)	95%	70%

Analysis of peak demand period

- Peak demand periods occurred during most expensive timeframe (8am-6pm) for all months in 2018
- Safe to assume peak demand charge (B) is $44.25/kW in June-September and $18.17/kW in other months



Lithium-ion battery or thermal energy storage

- Use demand data of Columbia University in 2018
- Energy rating
- Peak Shaving Profit

  for each year

thermal-ion battery	thermal battery
0	50
100	150
200	250
300	350
400	450
500	550
600	650

Lithium-ion battery or thermal energy storage

- Net present value (NPV)

➡ a financial metric that seeks to capture the total value of an investment opportunity

$$\text{Net Present Value (NPV)} = -C_0 + \frac{C_1}{(1+r)} + \frac{C_2}{(1+r)^2} + \cdots + \frac{C_T}{(1+r)^T}$$

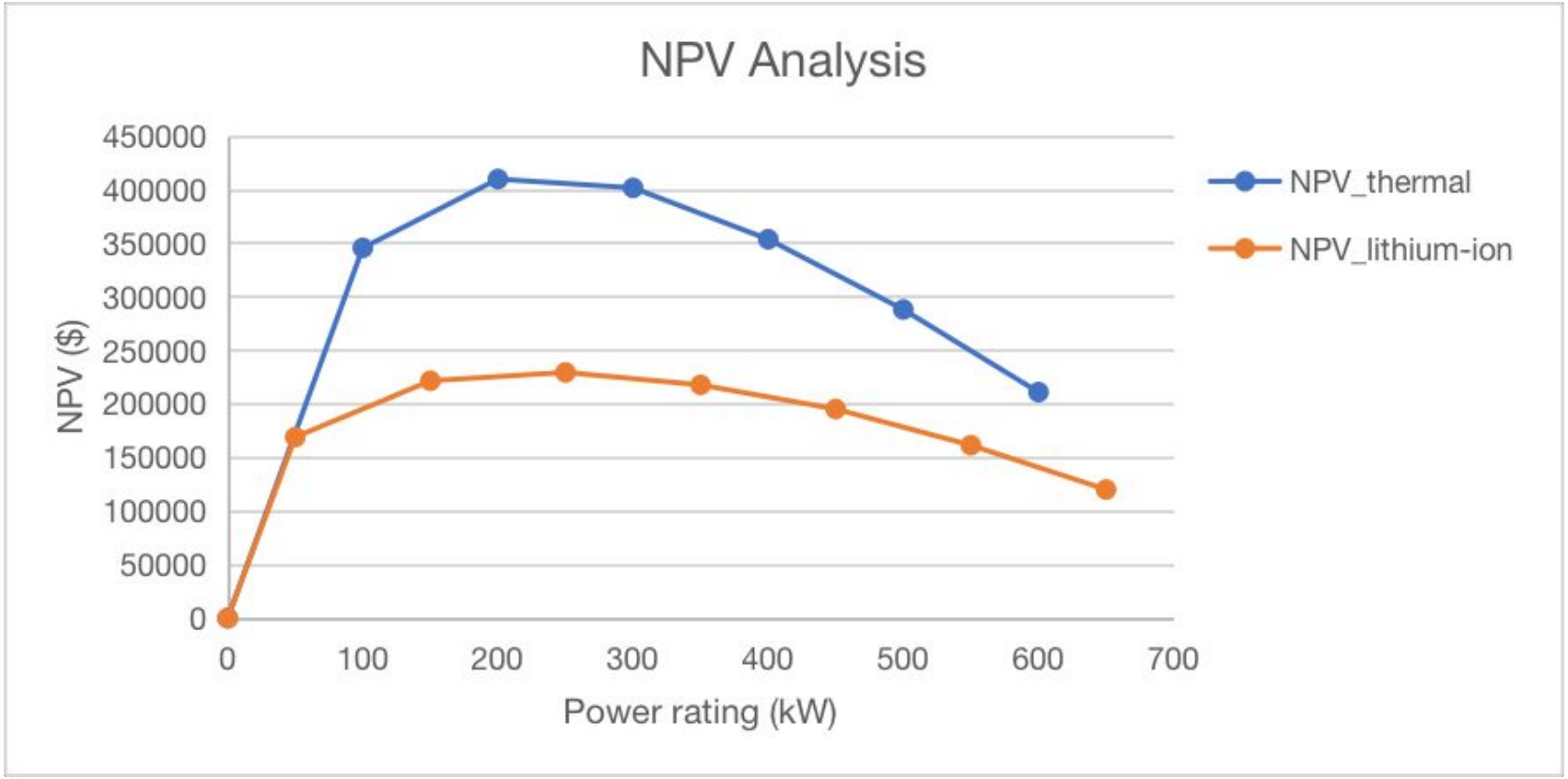
$-C_0$ = Initial Investment

C = Cash Flow

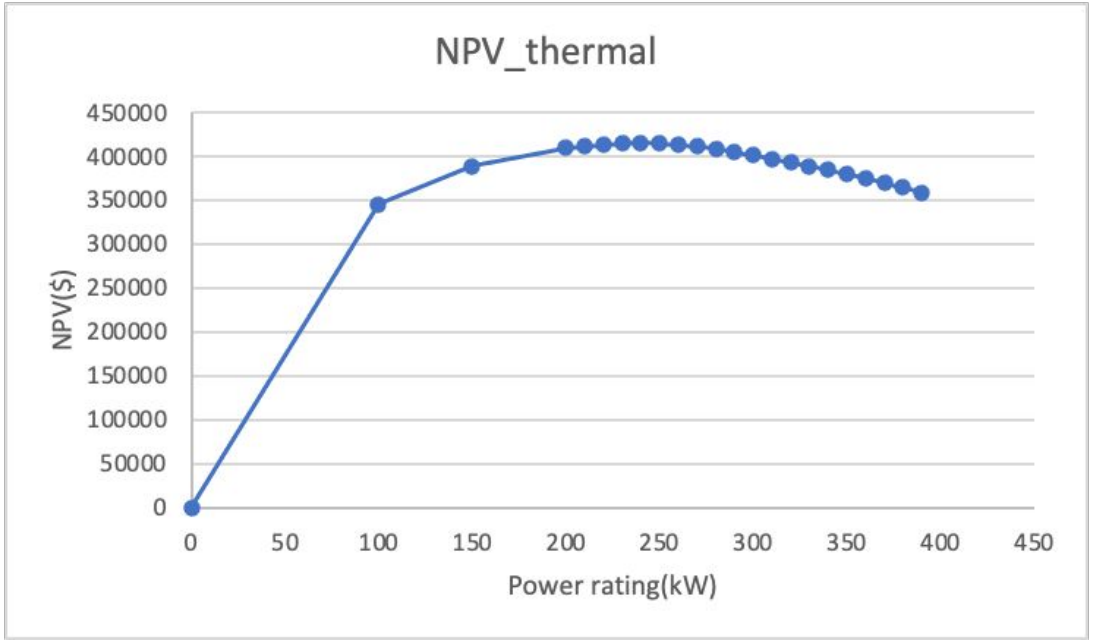
r = Discount Rate

T = Time

Lithium-ion battery or thermal energy storage



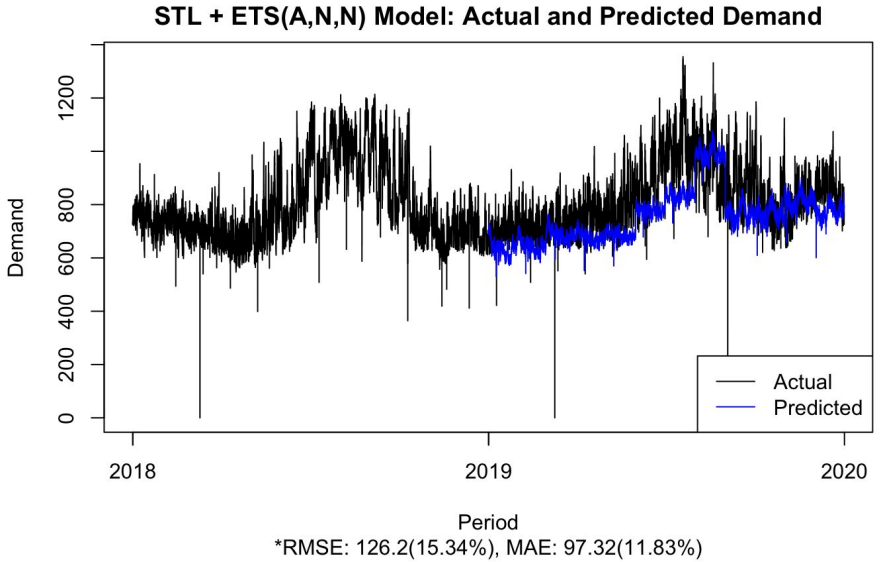
Find the optimal energy rating for 2018 peak shaving



- Thermal energy storage
- Energy rating: 240kW
- Investment: $264,000
- Market discount rate: 8%
- NPV: $415,758
- IRR: 37%

Forecasting

- Best forecast: STLF
- Root mean squared error ~126MW (15%) for 2019 forecasted demand
- Next steps:
  - Use forecast to determine how to operate battery in 2019 (and subsequent years)
  - Quantify the predicted cost savings vs. actual cost savings



Thank you!

Appendix

```
def run_optimization(D, P, battery, month):
    """
    Description: Code to run the optimization procedure

    Inputs:
        D, np.array: demand for each period
        P, float: battery power rating
        battery, str : battery option either "li-ion" or "thermal"
        month, int: month of year

    Output:
        Results of optimization
    """
    #Update variables depending on battery type
    if battery == 'li-ion':
        #battery efficiency
        eta = .95
        #Energy rating
        E = 4*P
    elif battery == 'thermal':
        #battery efficiency
        eta = .7
        #Energy rating
        E = 12*P
    else:
        print('optimizing without battery...')
        P=0
        eta = .95
        E = 0
```

- Code incorporates efficiency, energy rating, and power rating of each battery

Appendix

```
#Update peak charges depending on time of year:
if month in [6, 7, 8, 9]:
    B = 9.15 + 18.44 + 16.66
else:
    B = 4.21 + 13.96

#Define additional variables
cost_P = 300*P
cost_E = 200*E
#Cost of electricity
C = .13

# Decision Variables
#Storage discharge power
d = cp.Variable(len(D), nonneg=True)
# Storage charge power
q = cp.Variable(len(D), nonneg=True)
# Energy storged
e = cp.Variable(len(D), nonneg=True)
# peak demand
p = cp.Variable(nonneg=True)
```

- Update peak demand charge by month of optimization
- Define decision variables including storage discharge and charge power, energy stored, and peak demand

Appendix

```python
# Initialize an empty constraint set
con_set_1 = []

for t in range(len(D)):
    #Can't discharge more than the power rating of battery
    con_set_1.append(d[t] <= P)
    #Can't charge more than the power rating of battery
    con_set_1.append(q[t] <= P)
    # #Can't store more energy than capacity
    con_set_1.append(e[t] <= E)
    if t == 0:
        # state-of charge constraint 1 - assume battery is empty
        # before month starts
        con_set_1.append(e[t] == q[t]*eta - d[t]/eta)
    elif t == len(D)-1:
        # # peak demand identification
        con_set_1.append(p >= D[t] - d[t] + q[t])
    else:
        # # state-of charge constraint 1
        con_set_1.append(e[t] - e[t-1] == q[t]*eta - d[t]/eta)
        # # peak demand identification
        con_set_1.append(p >= D[t] - d[t] + q[t] + D[t+1] - d[t+1] + q[t+1])
```

- Define constraints such as discharge, charge, and storage capacity, state-of-charge, and peak demand identification

Appendix

```
#Define Objective - BP + sum over t
#mutlipy by 2 because p is only a half hour increment right now
obj = cp.Minimize(2*B*p + C*sum(D - d + q))

# Solve the problem
prob1 = cp.Problem(obj, con_set_1)
prob1.solve(solver = "GUROBI", reoptimize=True)
prob1.solve();
```

- Define objective and solve
- Use the solution to choose optimal battery power/energy rating and type