OPTIMIZING DIESEL GENERATOR DISPATCH FOR A MICROGRID WITH A SOLAR PV ARRAY

GLEN WOODWORTH

1. Introduction

Generator dispatch is the process of assigning load setpoints to generators in order to meet the electrical load on a microgrid. The process is complicated by adding solar photovolataic (PV) arrays to the microgrid as a source of power. The goal of this project is to optimize the dispatch of diesel generators, with respect to fuel consumption, for a week on a microgrid with a solar PV array, at an hourly resolution. In reality, dispatch decisions must rely on predictions about future load and future solar resource availability. For this project, the focus is on the optimization of the dispatch. Therefore, we reduce the complexity of the problem by making a few assumptions.

First, we assume we have perfect knowledge of the load profile as well as the solar resource availability for a week. Second, each diesel generator is assumed to have a linear fuel curve. Finally, we ignore the existence of an inverter, which would be necessary on a real microgrid to convert DC power from the solar PV array into AC for use on the microgrid. As a result of these assumptions, our problem will be stated as a linear programming problem (LP). After converting to standard form, we solve the problem using Phase One and Simplex Method.

In the remainder of this section, we rigorously define the problem as a LP problem. Section 2 covers the conversion of the LP problem into SF, Section 3 applies the the SF conversion method to a small example, Section 4 presents some of the MATLAB implementations I used, Section 5 presents plots of results, and Section 6 discusses the results.

1.1. **Problem.** Let $l \in \{1, 2, \ldots, L\}$ represent the timesteps and $k \in \{1, 2, \ldots, K\}$ represent the diesel generators. Define $i = l + L(k-1)$ where x_i is the load setpoint for generator k at timestep l. For each generator k of maximum capacity M_k define fuel curve $g_k : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ by

$$g_k(x) = 0.4234(M_k)^{-0.1012}x + 0.0940 \cdot (M_k)^{-0.2735}$$

where $g_k(x)$ is the fuel consumption of generator k in L/kWh at load setpoint x kW. This function g_k came from page 56 of the user manual at [1]. We can now define our objective:

$$\text{minimize} \quad z = \sum_{l=1}^{L} \sum_{k=1}^{K} g_k(x_i)$$

We now discuss the constraints. Let $D \in \mathbb{R}^L$ be the electrical demand, in kW. Define $S \in \mathbb{R}^L$ to be the normalized electrical output of the solar PV array, where $0 \leq S_l \leq 1$ is the ratio of the output of the array to its maximum capacity. Let r be the rated capacity of the solar PV array in kW. Then $S \cdot r$ is the electrical output of the solar PV array in kW.

Date: December 13, 2024.

We require that the sum of the output of diesel generators with the output of the solar PV array meets the demand exactly at each timestep l,

$$S_l \cdot r + \sum_{k=1}^{K} g_k(x_i) = D_l.$$

To ensure this is always possible, it is necessary to check that

(1) $D - S \cdot r \geq 0$ and
(2) $\sum_{k=1}^{K} M_k \geq D_l - S_l \cdot r$ for all $l \in \{1, 2, \ldots, L\}$.

The second condition was met by our data but the first condition forced preprocessing $D - S \cdot r$ using

$$D'_l = \max(D_l - S_l \cdot r, 0).$$

Our new equality constraints are

$$\sum_{k=1}^{K} x_i = D'_l,\ l \in \{1, 2, \ldots L\}.$$

For general inequality constraints, we have one for each generator at every timestep, that is, we have a total of $K \cdot L$ upper-bound inequlaity constraints, each of the following form

$$x_i \leq M_k.$$

Hence, we have a grand total of $m = L + KL$ general constraints. We can now state our linear program:

$$\begin{aligned} \text{minimize} \quad & z = \sum_{l=1}^{L} \sum_{k=1}^{K} g_k(x_i) \\ \text{subject to} \quad & \sum_{k=1}^{K} x_i = D'_l,\ l \in \{1, 2, \ldots L\} \\ & x_i \leq M_k.,\ l \in \{1, \ldots, L\}, k \in \{1, \ldots K\} \\ & x \geq 0. \end{aligned}$$

We now convert the linear program stated above into standard form (SF).

2. Standard Form

Note that the right hand side of every constraint is nonnegative by design. Thus, we have three tasks in this section:[1]

(1) add $K \cdot L$ slack variables to turn the general inequalities into equalities,
(2) rewrite objective z as $z - p = c^T x$ where p is some scalar, and
(3) write the resulting constraints in matrix vector form.

For (1), Define $j = K \cdot L + i$. For each x_i, we append a slack variable x_j to x to create a new vector of variables $x' \in \mathbb{R}^n$ where $n = 2 \cdot K \cdot L$. Now our general inequality constraints can be rewritten as the following equality constraints:

$$x'_i + x'_j = M_k,\ l \in \{1, \ldots, L\}, k \in \{1, \ldots, K\}.$$

[1]These steps were implemented in matlab functions available at the github repository for this project in the folder 'functions': https://github.com/gwoodworth/microgrid-optimization.

For (2), recall our objective function

$$z = \sum_{l=1}^{L}\sum_{k=1}^{K} g_k(x_i) = \sum_{l=1}^{L}\sum_{k=1}^{K} \left[0.4234(M_k)^{-0.1012}x_i + 0.0940 \cdot (M_k)^{-0.2735}\right].$$

Then for

$$p = \sum_{l=1}^{L}\sum_{k=1}^{K} 0.0940 \cdot (M_k)^{-0.2735},$$

we have

$$z - p = \sum_{l=1}^{L}\sum_{k=1}^{K} \left[0.4234(M_k)^{-0.1012}x_i\right].$$

Define $c \in \mathbb{R}^{K \cdot L}$ by

$$c_t = 0.0940 \cdot (M_k)^{-0.2735},\ 1 \leq t \leq K \cdot L$$

Thus, $z - p = c^T x$. Since we have added $K \cdot L$ slack variables, define block vector $c' \in \mathbb{R}^n$ by

$$c' = \begin{pmatrix} c \\ \mathbf{0} \end{pmatrix}$$

where $\mathbf{0}$ is a $K \cdot L$ vector of zeros. Then $z - p = c'^{\,T} x'$. Let $z' = z - p$ so that our new objective is

$$\text{minimize } \ z' = c'^{\,T} x'.$$

For (3), we must represent our updated constraints

$$\sum_{k=1}^{K} x'_i = D'_l,\ l \in \{1, 2, \ldots L\}$$
$$x'_i + x'_j = M_k,\ l \in \{1, \ldots, L\}, k \in \{1, \ldots, K\}.$$

as $Ax' = b$ where A is an $m \times n$ matrix and $b \in \mathbb{R}^m$. We construct A using blocks. To represent the demand constraint, we construct the following $L \times m$ block matrix by placing K identity matrices I_L side-by-side then padding the end with an $L \times (n/2)$ block of zeros:

$$I_{L \times n} = \begin{pmatrix} I_L & I_L & \ldots & I_L & 0_{L \times n/2} \end{pmatrix}$$

so that

$$I_{L \times m} x' = D'.$$

To represent the maximum capacity constraint, we must first represent the right-hand side as a vector. Define $M' \in \mathbb{R}^{n/2}$ by

$$M'_i = M_k$$

for $k \in \{1, \ldots, K\}$. For the left-hand side, we place two $I_{n/2}$ identity matrices side-by-side in a block matrix:

$$I_{n/2 \times n} = \begin{pmatrix} I_{n/2} & I_{n/2} \end{pmatrix}$$

so that

$$I_{n/2 \times n} x' = M'$$

is equivalent to the maximum capacity constraint. What remains is to define A and b. We do so using the following block matrix and block vector:

$$A = \begin{pmatrix} I_{L \times n} \\ I_{n/2 \times n} \end{pmatrix} \qquad \text{and} \qquad b = \begin{pmatrix} D' \\ M' \end{pmatrix}.$$

We can now write our linear program in matrix vector form. We have

$$\begin{aligned} \text{minimize} \quad & z' = c'^{\,T} x' \\ \text{subject to} \quad & Ax' = b \\ & x' \geq 0. \end{aligned}$$

3. A Small Example

For this example, we consider the case of three generators, with capacities 100 kW, 150 kW, and 250 kW over two hours with a 100 kW solar PV array. We follow the steps from Section 2 to put this example problem in SF, although the order is not followed exactly.

Let $K = 3$ where $M = (100, 150, 250)^T$, $L = 2$, $D = (500, 400)^T$, $S = (0.75, 1)^T$, and $r = 100$. Thus,

$$n = 2 \cdot K \cdot L = 2(3)(2) = 12 \qquad \text{and} \qquad m = L + K \cdot L = 2 + (3)(2) = 8.$$

First, we preprocess $D - S \cdot r$ to find D'. We have

$$D'_1 = \max(D_1 - S_1 \cdot r, 0) = \max(500 - (0.75 \cdot 100), 0) = \max(500 - 75, 0) = 425$$
$$D'_2 = \max(400 - 100, 0) = 300$$

so that $D' = (425, 300)$. Thus, using M and D' we construct b,

$$b = \begin{pmatrix} 425 \\ 300 \\ 100 \\ 100 \\ 150 \\ 150 \\ 250 \\ 250 \end{pmatrix}.$$

For A, we follow the construction using block matrices. We have

$$I_{L \times n} = (I_2 \; I_2 \; I_2 \; 0_{2\times 6})$$

and

$$I_{n/2 \times n} = (I_6 \; I_6)$$

so that

$$A = \begin{pmatrix} I_{2\times 12} \\ I_{6\times 12} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

What remains is to write the objective function as $z - p = c^T x$ so that we can convert it to $z' = c'^{\,T} x'$ where $x' \in \mathbb{R}^n$. In this case, it is more useful to cite the MATLAB functions I wrote for this task, **genSlopeYint** and **findc**. Only **findc** is shown here. For **genSlopeYint**, note that [slopes,yInt] =**genSlopeYint**(M) where M is K-vector of the

maximum capacities of the diesel generators, slopes is a K-vector containing the slopes of each g_K, and yInt is a K-vector containing the y-intercept for each g_K.

```
function [c,p] = findc(ratings,L)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FINDC Preprocessing function to produce c for use with simplex.
% Only c is necessary, but p might be useful at the end to get back to the
% actual value of the objective function (z=c^Tx+p).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    K = length(ratings);
    n = K*L;
    [slopes,yInt] = genSlopeYint(ratings);
    c = zeros(2*n,1); %2*n to account for the extra n slack variables
    p = zeros(2*n,1);
    for k=1:K
        for l=1:L
            i = l + L*(k-1);
            c(i) = slopes(k);
            p(i) = yInt(k);
        end
    end
    p = sum(p);
end
```

Then I ran the following code.

```
>> [newc,p] = findc([100,150,250],2)

newc =

    0.2657
    0.2657
    0.2550
    0.2550
    0.2421
    0.2421
         0
         0
         0
         0
         0
         0


p =

    0.1426
```

In this case, c' =newc from the above code session. Thus, our problem is

$$\begin{aligned} \text{minimize} \quad & z' = {c'}^T x' \\ \text{subject to} \quad & Ax' = b \\ & x' \geq 0. \end{aligned}$$

We will solve this example, among others, using a slightly modified version of **optimize** script (see Section 4) called **optimizeExample**. The modifications amount to simply changing the input variables to match this example, e.g., D, S, M, etc.

4. Implementation

The following script, called **optimize**, is my implementation of the methods discussed above.

```
% OPTIMIZE This is my matlab implementation
%load data, once per session
if exist('TTD','var') == 0
    TTD = readtimetable(...
        'data/electrical_load_generic_peak-500kW_1yr_dt-1hr.csv'...
        ); %Electrical demand as a timetable
end
if exist('TTS','var') == 0
    TTS = readtimetable(...
        'data/normalized_solar_production.csv'...
        ); %Normalized solar array output for a year
end
%Set parameters
ratings = [100,150,250]; % Ratings for diesel generators in kW,
%one for each generator
r = 250; %Total Solar PV array rating
L = 168; %number of timesteps

K = length(ratings); %number of diesel generators
n = K*L; %problem dimension,

m = L+K*L; %number of constraints after adding slacks
n = 2*n; %dimension after adding slacks

%Convert to standard form
D = TTD.ElectricalLoad_kW_(1:L); % Trim demand data
S = TTS.NormalizedProduction__(1:L); % Trim solar data
[c,p] = findc(ratings,L); % Find c for c^Tx and p, z=c^Tx+p
A = findA(K,L); % Find constraint matrix A
b = findb(D,S,ratings,r,K,L);% Find b for Ax=b

%Find an initial solution
```

```
xBFS = phaseone(A,b);
%run simplex
[x,z,iters] = sfsimplex(c,A,b,xBFS,false,10);

%Convert x and z back to original problem
x_og = b(L+1:m)-x(n/2+1:n);
z_og = z+sum(p);
```

```
>> optimize         % here I am running the code
ending: optimum found on iteration 920      %iterations of phaseone
phase 1 complete ... found basic feasible solution  %result of phaseone
ending: optimum found on iteration 1        %result of sfsimplex
```

After modifying the relevant input variables to match the small example from Section 3, I ran the following code.

```
>> optimizeExample %here I am running code
ending: optimum found on iteration 10
phase 1 complete ... found basic feasible solution
ending: optimum found on iteration 1
```

Notice that phaseone took 920 iterations in the former and 10 in the latter, while sfsimplex took 1 iteration in both cases. Regardless of L, my experimentation revealed that this is scenario seems to happen every time – phasone takes many iterations, but sfsimplex only takes 1. It appeared that the BFS from phaseone is always an optimal BFS.

To investigate this phenomenon, I made a minor modification to **optimize** by adding a check to see if the output from phaseone is exactly the same as the output of sfsimplex. I called this script **BFStest**. In **BFStest** the variable 'foo' is defined by

```
foo = all(x==xBFS);
```

where 'x' is the result of sfsimplex and 'xBFS' is the result of phaseone. That is, if 'foo = 1' then the output of phasone was an optimal BFS. I implemented the test in the following code, called **test**.

```
%%%%%
% TEST Script for testing if the output of phaseone is always an optimal BFS
%%%%%
%load data, once per session
if exist('TTD','var') == 0
    TTD = readtimetable(...
        'data/electrical_load_generic_peak-500kW_1yr_dt-1hr.csv'...
        ); %Electrical demand as a timetable
end
if exist('TTS','var') == 0
    TTS = readtimetable(...
        'data/normalized_solar_production.csv'...
        ); %Normalized solar array output for a year
end
```

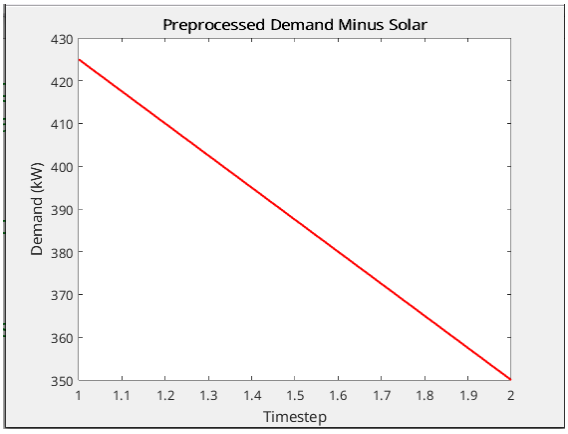
```
L=168;
outliers = [];
results = cell(L+1,4);
results{1,1} = 'L';
results{1,2} = 'T/F';
results{1,3} = 'initialIters';
results{1,4} = 'iters';
for l=1:L
   [L,foo,initialIters,iters] = bfstest(l,TTD,TTS);
   results{l+1,1} = L;
   results{l+1,2} = foo;
   results{l+1,3} = initialIters;
   results{l+1,4} = iters;
   if foo == false
       outliers=[outliers, L];
   end
end
```

The results of **test** are large, 168 lines in a matlab cell structure. We present a plot of them in the next section.

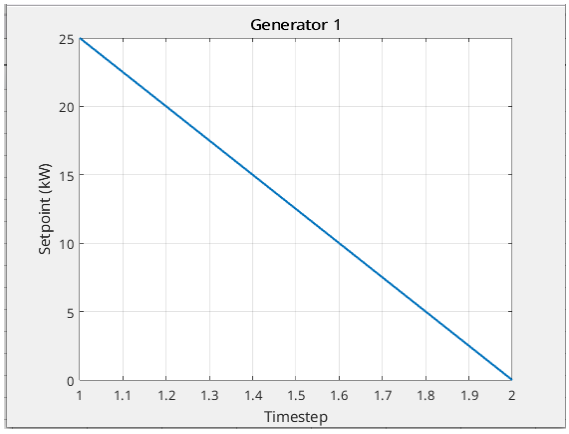
5. Results

This section contains the results from the small example, the test, and from the week long optimization experiment.

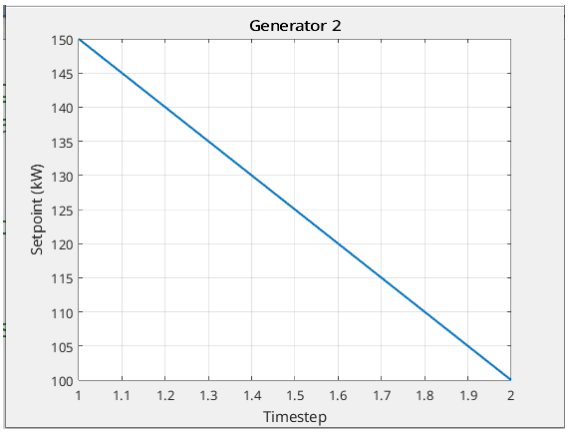
5.1. **Small Example.** Recall the parameters from the small example: $K = 3$, $M = (100, 150, 250)^T$, $L = 2$, $D = (500, 400)^T$, $S = (0.75, 1)^T$, and $r = 100$. The results are plotted in Figure 1.



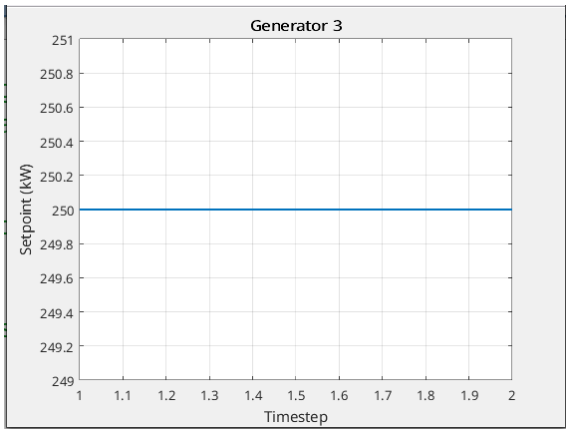
(A) Leftover demand



(B) Generator 1 setpoints



(C) Generator 2 setpoints



(D) Generator 3 setpoints

FIGURE 1. Plots of the leftover demand profile and the three generator setpoint profile for the small example.

5.2. **Test.** See Figure 5.2. What is not shown is that for every problem size from $L = 1$ to $L = 168$ the phaseone BFS was an optimal BFS.

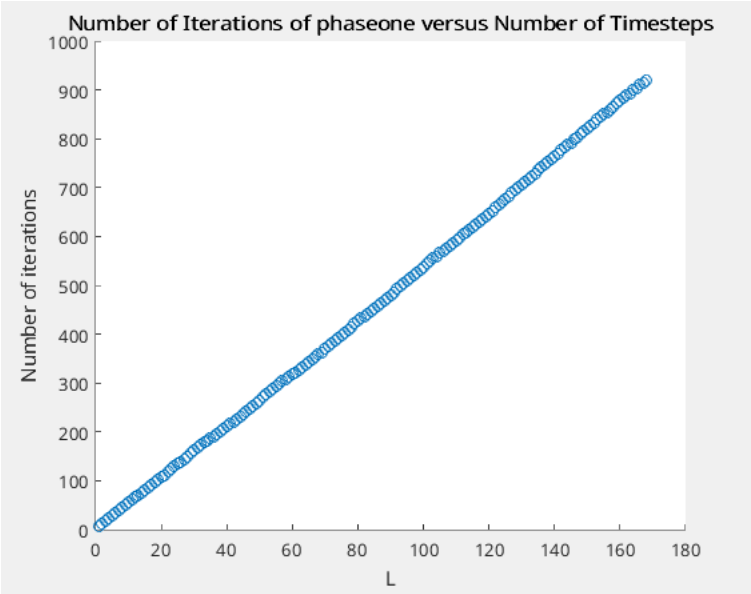
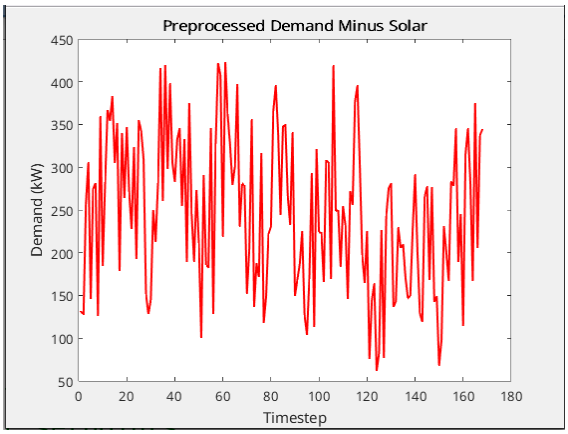
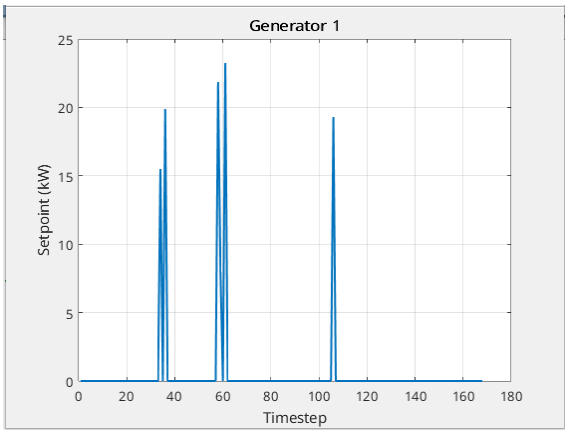


FIGURE 2. Iterations plot for phaseone

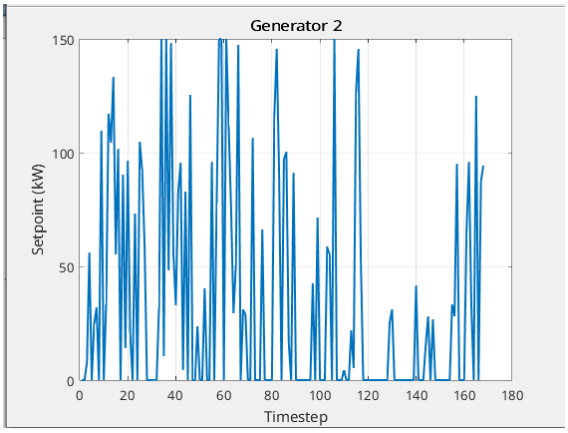
5.3. **Week.** The parameters for the week long experiment are: $K = 3$, $M = (100, 150, 250)^T$, $L = 168$, D and S are data from [1], and $r = 250$.



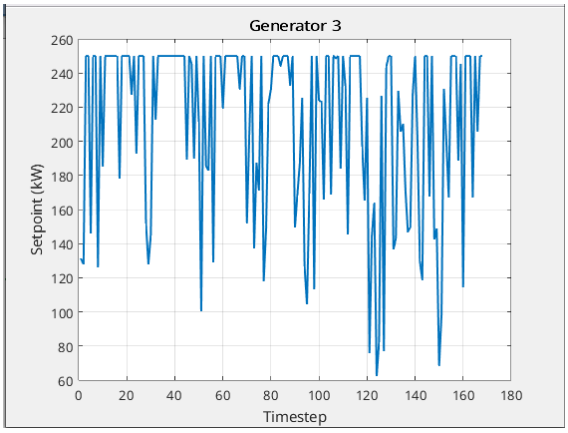
(A) Leftover demand



(B) Generator 1 setpoints



(C) Generator 2 setpoints



(D) Generator 3 setpoints

FIGURE 3. Plots of the leftover demand profile and the three generator setpoint profile for the week long experiment.

6. ANALYSIS

In the results for the test, notice that the number of iterations of phaseone increases linearly with respect to the number of timesteps, L (see Figure 5.2). Now, consider the results from the week experiment, see Figure 3. Notice that Generator 3 is near maximum capacity for the majority of the week, Generator 2 has is generally in the middle or bottom of its operating range, while Generator 1 is only used a few times when the demand cannot be met by Generator 1 and Generator 2. This makes sense because, based on our fuel curves, the larger a generator is, the more efficient it is.

Consider the results from the example in Figure 1. Similar behavior appears as from the week long experiment.[2] Generator 3 is at max capacity for both timesteps, while Generator 2 is at max at the first timestep while Generator 1 picks up the remaining load. Then at

[2]note that the y-axis for Generator 1 is capped at 25 kW while the maximum capacity for Generator 1 is 100 kW in this example.

the seccond timestep, Generator 1 is set to 0 while Generator 2 is set to 100, to pick up the remaining load Generator 3 could not meet.

This helps us understand the reason why the BFS from phaseone is always an optimal BFS. Each generator has a different efficiency. Ranked from most to least efficent: Generator 3, Generator 2, Generator 1. Since there is no penalty for operating a generator at maximum capacity or for starting and stopping a generator, there is only one way to meet the load in the most efficent manner at any given timestep – follow these steps:

(1) Use as much of Generator 3 as needed.
(2) If demand is leftover, use as much of Generator 2 as needed.
(3) If demand is leftover, use as much as Generator 1 as needed.

7. Conclusion

This project has shown something that an engineer would intuit – to use less fuel, use the most efficient generator as much as possible. The fact that our results match intuition is a good sign that our work was correct. What may not be not clear in that intuition is how to implement that logic in a meaningful way.

The linear programming approach presented here has a solid theoretical basis in optimization, of which it is only using a small piece of. That is, with small but important modifications, the approach used here could be made more realistic. One such modification could be to add a penalty when a generator operates below some minimum load. Or add a penalty for starting and stopping a generator repeatedly. Although the problem would no longer be linear programming, other optimization tools, such as POPDIP or other nonlinear optimization techniques could be used.

References

[1] Anthony Truelove. Pgmcpp. `https://github.com/gears1763-2/PGMcpp`, 2024.