EE 553 Spring 2015 Final Project Report

University of Washington

Darrell Ross

June 14, 2015

Contents

1 Introduction

This report covers an analysis of my code for a Mixed Integer Linear Program (MILP) written in *FICO Xpress's* Mosel language. The code is designed to find the optimal economic dispatch for a number of power generation units under a specific set of constraints.

I feel that the design of the project really requires a thorough report. If I were to report only on results, then I could conceivably not understand the code at all. It is important to me that I communicate clearly that I understand the code, how to write the constraints, and why they work.

Since there was a request for a concise report, I have broken my original report into a brief testing portion followed by extensive details in the appendices.

This report moves in an iterative story–like format. The program is started as basic as possible and tested. Then constraints are added one piece at a time and the model reverified in each step often using previous steps to test them. The sections will flow in the following order because this is the order that I wrote them in and it makes the most sense to me.

- **Load Generation Balance and Minimum Power and Maximum Power**
- **On and Off Status, Minimum Up Times, and Minimum Down Times**
- **No Load Cost, Startup Costs, and Reserve**
- **Three–Segment Cost Curves**

Each section will begin by explaining the variables needed for it followed by testing and verification.

1.1 Problem Statement

The problem statement was given as follows:

> *Write a Mixed Integer Linear Programming implementation of a Unit Commitment program using the student version of Xpress.*

1.2 Coding Style

A few notes about my coding style follow.

- Variables written in all upper–case letters are constants. Even if they are calculated constants, they are written in all upper–case letters.
- Variables written in all lower–case letters are decision variables.
- Variables have a array dimension based on basic inputs H for hours, P for prices, and G for generators. A variable with order "GxH" is a two–dimensional array that is G–by–H in size where G and H are defined in the data set.

2 Basic Economic Dispatch

Constraints used in this stage:

- Unit Power Limits
- Load Generation Balance

Required variables for this stage are shown in Table 1.

Variable	**Order**	**Type**	**Description**
H	1	integer	number of hours
G	1	integer	number of generators
HOURS	H	integer	indexes for each hour
GENS	G	integer	indexes for each generator
PMIN	G	real	minimum power values
PMAX	G	real	maximum power values
MARGINAL_PRICE	G	real	prices for one–segment cost curve
DEMAND	H	real	load values
unit_power	GxH	real	power supplied by each generator

Table 1: Variables used in the Basic Economic Dispatch program.

2.1 Testing Data: Practice Problem Set 3

Testing at this stage uses the sample data from Practice Problem Set #3 from class, shown in Table 2 and Table 3.

Unit	P_{min} **(MW)**	P_{max} **(MW)**	**Min Up Time (h)**	**Min Down Time (h)**	**No–Load Cost ($)**	**Marginal Cost ($/MWh)**	**Start–up Cost ($)**	**Initial Status**
A	**120**	**200**	**8**	**6**	**300**	**20**	**1500**	**-10**
B	**50**	**100**	**4**	**4**	**150**	**30**	**300**	**+5**
C	**30**	**50**	**2**	**3**	**40**	**200**	**100**	**+3**
D	**10**	**20**	**2**	**3**	**50**	**100**	**50**	**+3**

Table 2: Unit characteristics from Practice Problem Set 3.

Hour	1	2	3
Load	150	120	160

Table 3: Power demand per time period from Practice Problem Set 3.

2.2 Test: Match Practice Problem Set 3

With this basic information, it is clear that generation unit A can meet the demand every period and is the cheapest option. The input and output is shown below.

```
Input                                   Output
G:[4]                                   Cost Total: $8600
H:[3]                                   H1:Cost:$3000
PMIN:[120,50,30,10]                     P1:150MW; P2:0MW; P3:0MW; P4:0MW;
PMAX:[200,100,50,20]                    H2:Cost:$2400
MARGINAL_PRICE:[20,30,40,50]            P1:120MW; P2:0MW; P3:0MW; P4:0MW;
DEMAND:[150,120,160]                    H3:Cost:$3200
                                        P1:160MW; P2:0MW; P3:0MW; P4:0MW;
```

A total cost of $8600.00 is correct. In the Practice Problem Set 3 solutions, the final optimal result is $11000.00 but Practice Problem Set 3 included startup costs and no load costs, which, when accounted for, make up the difference.

2.3 Test: Match Increasing Loads

Leaving other inputs the same but increasing the loads so that each hour would need to use the next generator to meet its demand, the program correctly utilizes the generators as shown in the following results.

```
G:[4]                                   Cost Total: $30000
H:[4]                                   H1:Cost:$4000
PMIN:[120,50,30,10]                     P1:200MW; P2:0MW; P3:0MW; P4:0MW;
PMAX:[200,100,50,20]                    H2:Cost:$7000
MARGINAL_PRICE:[20,30,40,50]            P1:200MW; P2:100MW; P3:0MW; P4:0MW;
DEMAND:[200,300,350,370]                H3:Cost:$9000
                                        P1:200MW; P2:100MW; P3:50MW; P4:0MW;
                                        H4:Cost:$10000
                                        P1:200MW; P2:100MW; P3:50MW; P4:20MW;
```

This makes sense because the costs increase in the order *A, B, C, D*.

2.4 Test: Out of Bounds

To verify the program fails at the right points. Load values exceeding available supply were also tested and resulted in the program correctly failing to converge. Failure to converge results in zeros for all outputs as follows:

```
G:[4]                                   Cost Total: $0
H:[4]                                   H1:Cost:$0
PMIN:[120,50,30,10]                     P1:0MW; P2:0MW; P3:0MW; P4:0MW;
PMAX:[200,100,50,20]                    H2:Cost:$0
MARGINAL_PRICE:[20,30,40,50]            P1:0MW; P2:0MW; P3:0MW; P4:0MW;
DEMAND:[100,200,600]                    H3:Cost:$0
                                        P1:0MW; P2:0MW; P3:0MW; P4:0MW;
```

3 Dispatch with On and Off Status and Minimum Times

Constraints added in this stage:

- On and Off Status
- Minimum Up Time and Minimum Down Time

Required variables for this stage are shown in Table 4. New variables are highlighted in yellow.

Variable	**Order**	**Type**	**Description**
H	1	integer	number of hours
G	1	integer	number of generators
HOURS	H	integer	indexes for each hour
GENS	G	integer	indexes for each generator
PMIN	G	real	minimum power values
PMAX	G	real	maximum power values
MARGINAL_PRICE	G	real	prices for one–segment cost curve
DEMAND	H	real	load values
unit_power	GxH	real	power supplied by each generator
M	1	integer	large number for clever constraints
MIN_UP	G	integer	minimum up time for generators
MIN_DOWN	G	integer	minimum down time for generators
STATUS_INIT	G	integer	initial up and down time for generators
unit_on	GxH	binary	generators on or off
actual_power	GxH	real	power supplied by each generator that is on

Table 4: Variables used in the Dispatch with On and Off Status and Minimum Times. New variables are highlighted yellow.

3.1 Test: Match Practice Problem Set 3

When introducing new constraints, the first test is always to see if the function will match the previous version given input which does not effect change with the new constraints. Setting *MIN_UP* and *MIN_DOWN* to 1 hour for each generator and setting *STATUS_INIT* to -1 (down for one hour) for each generator eliminates the effect of both *On and Off Status* and *Minimum Times*.

```
G:[4]
H:[3]
PMIN:[120,50,30,10]
PMAX:[200,100,50,20]
MARGINAL_PRICE:[20,30,40,50]
DEMAND:[150,120,160]
MIN_UP:[1,1,1,1]
MIN_DOWN:[1,1,1,1]
STATUS_INIT:[-1,-1,-1,-1]
```

```
Cost Total: $8600
H1:Cost:$3000
P1:150MW; P2:0MW; P3:0MW; P4:0MW;
H2:Cost:$2400
P1:120MW; P2:0MW; P3:0MW; P4:0MW;
H3:Cost:$3200
P1:160MW; P2:0MW; P3:0MW; P4:0MW;
```

3.2 Test: On and Off Status in Practice Problem Set 3

For the following tests of *On and Off Status*, the following inputs are assumed and only the *MIN_UP*, *MIN_DOWN*, and *STATUS_INIT* are changed each time.

```
G:[4]
H:[3]
PMIN:[120,50,30,10]
PMAX:[200,100,50,20]
MARGINAL_PRICE:[20,30,40,50]
DEMAND:[150,120,160]
```

Printing the on and off status in binary numbers for each time period with inputs on the left and results on the right, a match to the Basic Economic Dispatch is shown here:

```
MIN_UP:[1,1,1,1]                                1000
MIN_DOWN:[1,1,1,1]                              1000
STATUS_INIT:[-1,-1,-1,-1]                       1000
```

3.3 Test: On and Off Status of Homework 4, Problem 2

The data from Homework 4, Problem 2 is shown in Table 5 and Table 6.

Unit	P_{min} **(MW)**	P_{max} **(MW)**	**Min Up Time (h)**	**Min Down Time (h)**	**No–Load Cost (\$)**	**Marginal Cost (\$/MWh)**	**Start–up Cost (\$)**	**Initial Status**
A	**180**	**250**	**3**	**3**	**0**	**10**	**1000**	**-5**
B	**70**	**100**	**2**	**2**	**0**	**12**	**600**	**+3**
C	**10**	**50**	**1**	**1**	**0**	**20**	**150**	**+3**

Table 5: Unit characteristics from Homework 4, Problem 2.

Hour	1	2	3
Load	320	250	260

Table 6: Power demand per time period from Homework 4, Problem 2.

Testing *On and Off Status* with this data produces results which match one possible optimal path solution, that of part e from the homework.

```
H:[3]                                   Cost Total: $8540
G:[3]                                   H1:Cost:$3340
PMIN:[180,70,10]                        P1:250MW; P2:70MW; P3:0MW;
PMAX:[250,100,50]                       H2:Cost:$2500
MARGINAL_PRICE:[10,12,20]               P1:250MW; P2:0MW; P3:0MW;
DEMAND:[320,250,260]                    H3:Cost:$2700
MIN_UP:[3,2,1]                          P1:250MW; P2:0MW; P3:10MW;
MIN_DOWN:[3,2,1]
STATUS_INIT:[-5,3,3]                    110
                                        100
                                        101
```

The cost for this solution in the Homework was $9690 but that included $1150 in startup costs which makes up the difference here.

3.4 Test: Minimum Up Time

With *On and Off Status* verified as functional, I can test Minimum Up Time more thoroughly. This is done using Practice Problem Set 3 since the results are more predictable and easier to see.

Initial Problem Results
Restating the initial results with only the new inputs, we had the following:

```
MIN_UP:[1,1,1,1]                        1000
MIN_DOWN:[1,1,1,1]                      1000
STATUS_INIT:[-1,-1,-1,-1]               1000
```

Up Time Boundary
Setting the Minimum Up Time of Unit 4 to 4 hours and with an initial status of up for only 1 hour should force it to be on for three hours and effects other generators which will need to compensate as shown here in the results:

```
MIN_UP:[1,1,1,4]                        1001
MIN_DOWN:[1,1,1,1]                      0101
STATUS_INIT:[-1,-1,1,1]                 1001
```

Setting the Minimum Up Time of Unit 3 to 2 hours and giving it an initial up time of 1 hour should force it to be on for at least the first hour:

```
MIN_UP:[1,1,2,4]                        0111
MIN_DOWN:[1,1,1,1]                      0101
STATUS_INIT:[-1,-1,1,1]                 1001
```

Since Units 3 and 4 were forced to be on in the first hour, their minimum output sums to 40MW which is only 110MW below the demand for period 1. Since Unit 1 has a PMIN of 120MW, it could not be used and that is reflected in the output. If I set this for all generators, it causes the problem to fail to converge because the sum of the minimum power outputs is 210MW which exceeds the demand in all three hours.

3.5 Test: Minimum Down Time

Minimum Down Time is very similar in form and theory to Minimum Up Time but it is possible that the constraints have bugs so it is good to perform more tests.

Down Time Boundary
For Minimum Down Time, I set Unit A minimum down time to 2 hours with an initial down status of 1 hour. This means it cannot be turned on until Period 2 and the results show it is working:

```
MIN_UP:[1,1,1,1]                          0110
MIN_DOWN:[2,1,1,1]                        1000
STATUS_INIT:[-1,-1,-1,-1]                 1000
```

3.6 Test: Minimum Up Time and Minimum Down Time

Combining the two can lead to interesting tests. Starting with the Minimum Up Time test where Unit 4 was forced to be on the whole time:

```
MIN_UP:[1,1,1,4]                          1001
MIN_DOWN:[1,1,1,1]                        0101
STATUS_INIT:[-1,-1,1,1]                   1001
```

Since Unit 1 turned off in Period 2, this is a good spot to introduce a Minimum Down Time constraint to test them together. Setting a Minimum Down Time of 2 hours for Unit 1 should create the same results as above except that Unit 1 will have to turn off for Period 3:

```
MIN_UP:[1,1,1,4]                          0111
MIN_DOWN:[2,1,1,1]                        0101
STATUS_INIT:[1,-1,-1,1]                   1001
```

Even though this did not do what I expected based purely on on/off statuses, reviewing the total costs tells me why. If Unit 1 was on, then the total cost would have been $12800 but with the optimal result the program found, the total cost for the above results was $12600.

I can force the results I expected by adding a Minimum Up Time of 2 hours to Unit A with an initial status of 1 hour up. This produces what I thought would be the transition:

```
MIN_UP:[2,1,1,4]                          1001
MIN_DOWN:[2,1,1,1]                        0101
STATUS_INIT:[1,-1,-1,1]                   0111
```

And the total cost is $12800. This is an example of how the constraints and complexity of the problem can easily get complicated quickly. In this case, my program found the right solution and I am satisfied that it is functional so far.

4 Dispatch with No Load Cost, Startup Costs, and Reserve

Constraints added in this stage:

- No Load Cost
- Startup Costs
- Reserve

Required variables for this stage are shown in Table 7. New variables are highlighted in yellow.

Variable	**Order**	**Type**	**Description**
H	1	integer	number of hours
G	1	integer	number of generators
HOURS	H	integer	indexes for each hour
GENS	G	integer	indexes for each generator
PMIN	G	real	minimum power values
PMAX	G	real	maximum power values
MARGINAL_PRICE	G	real	prices for one–segment cost curve
DEMAND	H	real	load values
unit_power	GxH	real	power supplied by each generator
M	1	integer	large number for clever constraints
MIN_UP	G	integer	minimum up time for generators
MIN_DOWN	G	integer	minimum down time for generators
STATUS_INIT	G	integer	initial up and down time for generators
STATUS_INIT_UP	G	integer	calculated from STATUS_INIT
STATUS_INIT_DOWN	G	integer	calculated from STATUS_INIT
UNITS_INIT	G	integer	calculated from STATUS_INIT
unit_on	GxH	binary	generators on or off
actual_power	GxH	real	power supplied by each generator that is on
NO_LOAD_COST	G	real	no load cost for each generator
STARTUP_COST	G	real	startup cost for each generator
RESERVE_PERCENT	1	real	percentage of demand to hold in reserve
unit_started	GxH	binary	generator started

Table 7: Variables used with the Dispatch with No Load Cost, Startup Costs, and Reserve program. New variables are highlighted yellow.

4.1 Test: Match Practice Problem Set 3

To make sure the program matches the solution for Practice Problem Set 3 from Section 3.2, I tested with inputs which did not effect the results. Notice that the *unit_started* calculation is clearly working. Unit A started in Period 1.

```
H:[3]
G:[4]
PMIN:[120,50,30,10]
PMAX:[200,100,50,20]
MARGINAL_PRICE:[20,30,40,50]
DEMAND:[150,120,160]
MIN_UP:[3,2,1]
MIN_DOWN:[3,2,1]
STATUS_INIT:[-5,3,3]
NO_LOAD_COST:[0,0,0]
STARTUP_COST:[0,0,0]
RESERVE_PERCENT:[0.0]
```

```
Cost Total: $8600
H1:Cost:$3000
P1:150MW; P2:0MW; P3:0MW; P4:0MW;
H2:Cost:$2400
P1:120MW; P2:0MW; P3:0MW; P4:0MW;
H3:Cost:$3200
P1:160MW; P2:0MW; P3:0MW; P4:0MW;

unit_on
0110(UNITS_INIT)
---
1000
1000
1000

unit_started
1000
0000
0000
```

4.2 Test: Match Homework 4, Problem 2

To make sure the program matches the solution for Homework 4, Problem 2 from Section 3.3, I tested with inputs which did not effect the results. Notice in the results that the *unit_started* calculation is clearly working. Unit A started in Period 1 and Unit C started in Period 3.

```
H:[3]
G:[3]
PMIN:[180,70,10]
PMAX:[250,100,50]
MARGINAL_PRICE:[10,12,20]
DEMAND:[320,250,260]
MIN_UP:[3,2,1]
MIN_DOWN:[3,2,1]
STATUS_INIT:[-5,3,3]
NO_LOAD_COST:[0,0,0]
STARTUP_COST:[0,0,0]
RESERVE_PERCENT:[0.0]
```

```
Cost Total: $8540
H1:Cost:$3340
H2:Cost:$2500
H3:Cost:$2700

unit_on
011(UNITS_INIT)
---
110
100
101

unit_started
100
000
001
```

4.3 Test: No Load Cost

No Load Cost is very simple to test. The cost is added once for each generator each hour that it is on. The previous test showed Unit A on for 3 hours, Unit B on for 1 hour, and Unit C on for 1 hour. By setting the *NO_LOAD_COST* to easy numbers, I can calculate what the differences should be and they work out.

```
NO_LOAD_COST: [5,6,7]
STARTUP_COST: [0,0,0]
RESERVE_PERCENT: [0.0]
```

```
Cost Total: $8568 = 8540+28
H1:Cost:$3351     = 3340+11
H2:Cost:$2505     = 2500+5
H3:Cost:$2712     = 2700+12
```

4.4 Test 3: Startup Cost

To test Startup Cost, I can compare to my results from Homework 4, Problem 2, Part f. The results match the solutions to the homework.[1]

```
H:[3]
G:[3]
PMIN:[180,70,10]
PMAX:[250,100,50]
MARGINAL_PRICE:[10,12,20]
DEMAND:[320,250,260]
MIN_UP:[3,2,1]
MIN_DOWN:[3,2,1]
STATUS_INIT:[-5,3,3]
NO_LOAD_COST:[0,0,0]
STARTUP_COST:[1000,600,150]
RESERVE_PERCENT:[0.0]
```

```
Cost Total: $9690
H1:Cost:$4340
P1:250MW; P2:70MW; P3:0MW;
H2:Cost:$2500
P1:250MW; P2:0MW; P3:0MW;
H3:Cost:$2850
P1:250MW; P2:0MW; P3:10MW;

110
100
101
```

4.5 Test 4: Reserve Percent

This test gets tricky because the reserve changes depending on the demand. Boundaries are the easiest to test.

100% Reserve
Setting the Reserve requirement to 100% causes a failure to solve. This is expected because it increases the demand requirement beyond the level that the generators can provide for.

```
NO_LOAD_COST:[0,0,0]
STARTUP_COST:[1000,600,150]
RESERVE_PERCENT:[1.0]
```

```
Cost Total: $0
H1:Cost:$0
P1:0MW; P2:0MW; P3:0MW;
H2:Cost:$0
P1:0MW; P2:0MW; P3:0MW;
H3:Cost:$0
P1:0MW; P2:0MW; P3:0MW;
```

[1] I was very excited when I got this far as well.

20% Reserve
Setting the Reserve requirement to 20% means the generators must have 64MW, 50MW, and 52MW for each period in reserve. This should cause the cost to go up because extra units will need brought up to handle the extra reserve requirement. The results to do not disappoint.

```
H:[3]
G:[3]
PMIN:[180,70,10]
PMAX:[250,100,50]
MARGINAL_PRICE:[10,12,20]
DEMAND:[320,250,260]
MIN_UP:[3,2,1]
MIN_DOWN:[3,2,1]
STATUS_INIT:[-5,3,3]
NO_LOAD_COST:[0,0,0]
STARTUP_COST:[1000,600,150]
RESERVE_PERCENT:[0.2]
```

```
Cost Total: $9820
H1:Cost:$4440
P1:240MW; P2:70MW; P3:10MW;
H2:Cost:$2640
P1:180MW; P2:70MW; P3:0MW;
H3:Cost:$2740
P1:190MW; P2:70MW; P3:0MW;

unit_on
011(UNITS_INIT)
---
111
110
110

unit_started
100
000
000
```

These results make sense.

- In Period 1, Unit 3 had to be brought online in order to meet reserve constraints which meant reducing Unit 1 by Unit 3's minimum power since Unit 2 was already at minimum power.
- In Period 2, Unit 1 could not meet the 50MW reserve so the minimum power had to be provided by Unit 2.
- In Period 3, the situation was the same as Period 2.

5 Dispatch with Three–Segment Cost Curves

Constraints added in this stage:

- Three–Segment Cost Curves

Required variables for this stage are shown in Table 8. New variables are highlighted in yellow.

Variable	**Order**	**Type**	**Description**
H	1	integer	number of hours
G	1	integer	number of generators
HOURS	H	integer	indexes for each hour
GENS	G	integer	indexes for each generator
PMIN	G	real	minimum power values
PMAX	G	real	maximum power values
MARGINAL_PRICE	G	real	prices for one–segment cost curve
DEMAND	H	real	load values
unit_power	GxH	real	power supplied by each generator
M	1	integer	large number for clever constraints
MIN_UP	G	integer	minimum up time for generators
MIN_DOWN	G	integer	minimum down time for generators
STATUS_INIT	G	integer	initial up and down time for generators
STATUS_INIT_UP	G	integer	calculated from STATUS_INIT
STATUS_INIT_DOWN	G	integer	calculated from STATUS_INIT
UNITS_INIT	G	integer	calculated from STATUS_INIT
unit_on	GxH	binary	generators on or off
actual_power	GxH	real	power supplied by each generator that is on
NO_LOAD_COST	G	real	no load cost for each generator
STARTUP_COST	G	real	startup cost for each generator
RESERVE_PERCENT	1	real	percentage of demand to hold in reserve
unit_started	GxH	binary	generator started
P	1	integer	number of segments in cost curve
PRICES	P	integer	indexes for each cost curve
split_power	PxGxH	real	power for each segment of the generator

Table 8: Variables used with the Dispatch with Three–Segment Cost Curves program. New variables are highlighted yellow.

5.1 Test: Match Practice Problem Set 3

The following inputs were used and the results matched. It is useful to look at the *split_power* results here to see how it is working.

```
H:[3]
G:[4]                                            Cost Total: $8600
PMIN:[120,50,30,10]                              P1:150=120+26.6667+3.33333+0
PMAX:[200,100,50,20]                             P2:50=50+0+0+0
MARGINAL_PRICE:[20,30,40,50]                     P3:30=30+0+0+0
DEMAND:[150,120,160]                             P4:10=10+0+0+0
MIN_UP:[3,2,1]
MIN_DOWN:[3,2,1]                                 P1:120=120+0+0+0
STATUS_INIT:[-5,3,3]                             P2:50=50+0+0+0
NO_LOAD_COST:[0,0,0]                             P3:30=30+0+0+0
STARTUP_COST:[0,0,0]                             P4:10=10+0+0+0
RESERVE_PERCENT:[0.0]
P:[3]                                            P1:160=120+0+26.6667+13.3333
PRICE:[20,30,40,50,                              P2:50=50+0+0+0
       20,30,40,50,                              P3:30=30+0+0+0
       20,30,40,50]                              P4:10=10+0+0+0
```

In Period 1, Unit 1 is supplying 150MW where the minimum is 120MW and then the remaining 80MW is split into three sections of 26.666MW each. Since 30MW is needed, the first price is used up completely at 26.66MW and the second is used partially at 3.33MW.

5.2 Test: Verify Three–Segment

To verify the soundness of the design, I created a simple input file which produces the situation present in Homework 8, Problem 3. If the constraints were written correctly, then this should produce values that match the Running Costs for that homework.

```
H:[6]                                            Cost Total: $22812
G:[1]                                            H1:Cost:$2496
PMIN:[200]                                       H2:Cost:$2496
PMAX:[500]                                       H3:Cost:$3762
MARGINAL_PRICE:[0]                               H4:Cost:$5088
DEMAND:[200,200,300,400,500,200]                 H5:Cost:$6474
MIN_UP:[1]                                       H6:Cost:$2496
MIN_DOWN:[1]                                     three-segment
STATUS_INIT:[-1]                                 P1:200=200+0+0+0
NO_LOAD_COST:[2496]                              P1:200=200+0+0+0
STARTUP_COST:[0]                                 P1:300=200+100+0+0
RESERVE_PERCENT:[0.0]                            P1:400=200+100+100+0
P:[3]                                            P1:500=200+100+100+100
PRICE:[12.66,13.26,13.86]                        P1:200=200+0+0+0
```

Since there is only one generator, I left the power generation results out of the results display because they have to match the demand. The hourly costs match my calculations from Homework 8, Problem 3.

6 Comprehensive Test

After testing each additional option in a vacuum, there are some limited tests that can be accomplished on the large sample data set. I cannot verify the answers precisely without doing the problem first by hand and even then, I am likely to make mistakes by hand. However, I am confident that my code works so that solving large problems will produce valid results.

I can also verify differences. For example, when No Load Cost is introduced, I should see an increase similar to that shown in the test. Or when Hot Start is introduced, I should see the price difference appear between \$0 and \$50 for a single Hot Start.

For this test, I used the sample project data provided and shown here in Table 9 and Table 10.

Unit	**P_{min} (MW)**	**P_{max} (MW)**	**Min Up Time (h)**	**Min Down Time (h)**	**No–Load Cost (\$)**	**Marginal Cost (\$/MWh)**	**Hot Start–up Cost (\$)**	**Cold Start–up Cost (\$)**	**Initial State**
1	25	100	1	1	0	15.4500	50.00	500.00	2
2	50	150	1	1	0	17.3467	60.00	700.00	-3
3	60	200	3	3	0	26.9000	100.00	800.00	-5
4	0	250	2	1	0	36.9000	90.00	900.00	-7

Table 9: Unit characteristics from the sample project data.

Hour	1	2	3	4	5	6
Load	250	200	390	250	130	300

Table 10: Power demand per time period from the sample project data.

6.1 Test: Match All Iterations of the Program

First, I ran the program with inputs based on the sample data which would produce results that could match the program as it appeared in *Dispath with On and Off Status and Minimum Times* and *Dispath with No Load Cost, Startup Costs, and Reserve* and *Dispath with Three–Segment Cost Curves.*

Notice the inputs had to be modified here so that all three programs would produce the same results and it worked.

```
H:[6]
G:[4]
P:[3]
PMIN:[25,50,60,0]
PMAX:[100,150,200,250]
MARGINAL_PRICE:[15.45,17.3467,26.9,36.9]
DEMAND:[250,200,390,250,130,300]
MIN_UP:[1,1,3,2]
MIN_DOWN:[1,2,2,1]
STATUS_INIT:[2,-3,-5,-7]
NO_LOAD_COST:[0,0,0,0]
STARTUP_COST:[0,0,0,0]
RESERVE_PERCENT:[0.0]
PRICE:[15.45,17.3467,26.9,36.9,
       15.45,17.3467,26.9,36.9,
       15.45,17.3467,26.9,36.9]
```

```
Cost Total: $28342.9
H1:Cost:$4720.2
P1:100MW; P2:90MW; P3:60MW; P4:0MW;
H2:Cost:$3871.84
P1:90MW; P2:50MW; P3:60MW; P4:0MW;
H3:Cost:$7913
P1:100MW; P2:150MW; P3:140MW; P4:0MW;
H4:Cost:$4147
P1:100MW; P2:150MW; P3:0MW; P4:0MW;
H5:Cost:$2103.34
P1:80MW; P2:50MW; P3:0MW; P4:0MW;
H6:Cost:$5587.54
P1:100MW; P2:140MW; P3:60MW; P4:0MW;

unit_on
1000(UNITS_INIT)
---
1111
1111
1111
1101
1101
1111

unit_started
0111
0000
0000
0000
0000
0010
```

There is a somewhat odd bit worth noting here. In Period 1, Unit 1 is set to 0MW but is also set to ON. This is because it has a minimum power of 0MW so it is possible to set it to ON with no power production.

Put in actual Startup Costs

With the Startup Costs updated from the table, the cost should increase and it does. Also, because the later generators now cost so much more to turn on, notice that the program doesn't utilize them as much as it did in the first run.

```
Cost Total: $30475.7
H1:Cost:$4847
P1:100MW; P2:150MW; P3:0MW; P4:0MW;
H2:Cost:$3279.67
P1:100MW; P2:100MW; P3:0MW; P4:0MW;
H3:Cost:$8713
P1:100MW; P2:150MW; P3:140MW; P4:0MW;
H4:Cost:$4720.2
P1:100MW; P2:90MW; P3:60MW; P4:0MW;
H5:Cost:$2828.27
P1:0MW; P2:70MW; P3:60MW; P4:0MW;
H6:Cost:$6087.54
P1:100MW; P2:140MW; P3:60MW; P4:0MW;

unit_on
1000(UNITS_INIT)                                 unit_started
---                                              0100
1100                                             0000
1100                                             0010
1110                                             0000
1110                                             0000
0110                                             1000
1110
```

Add Price Differences

Adding in price differences means that each generator will cost more for more higher power. This should result in the program utilizing lower power supply first since lower costs less. I can exaggerate this effect by making the upper third of the power for each generation unit cost a very large amount.

```
PRICE:[15,17,26,36,                              Cost Total: $34746.7
       20,25,30,40,                              H1:Cost:$6518.33
       30,35,40,50]                              P1:75MW; P2:115MW; P3:60MW; P4:0MW;
                                                 H2:Cost:$4760
                                                 P1:56.6667MW; P2:83.3333MW; P3:60MW; P4:0MW;
                                                 H3:Cost:$9123.33
                                                 P1:100MW; P2:136.667MW; P3:153.333MW; P4:0MW;
                                                 H4:Cost:$5018.33
                                                 P1:75MW; P2:115MW; P3:60MW; P4:0MW;
                                                 H5:Cost:$2110
                                                 P1:50MW; P2:80MW; P3:0MW; P4:0MW;
                                                 H6:Cost:$7216.67
                                                 P1:100MW; P2:150MW; P3:-1.42109e-013MW; P4:50MW;
```

The three–segment cost curves starting at near the original price definitely end up costing more.

7 A Note About Hot Start

I originally formulated a Hot Start solution and I still feel the constraints are valid but I am quite burnout out on writing this report.

Hot Start wasn't too bad to formulate because I already had the *UNIT_STARTED* data and the only pattern search for my Hot Start algorithm, which uses a 1–hour hot–start and everything else being a cold–start, is to look for the pattern 101. Since *UNIT_STARTED* is the 01 of that pattern, I needed only to look for the pattern where u(t)=1 and UNIT_STARTED(t+2)=1.

In the interest of my own sanity and spending more time with my family though, I did not complete testing of this feature. I assure you that I did get it working and you can verify that I formulated my constraints correctly as I have included them just below.

I thought it worth including this brief bit just in case it would net me some extra credit!

```
forall(g in GENS, h in HOURS|h=1) unit_hot_started(g,h)  >= unit_started(g,h)-(1-DOWN_FOR_ONE_HOUR(g))
forall(g in GENS, h in HOURS|h=2) unit_hot_started(g,h)  >= unit_started(g,h)-(1-UNITS_INIT(g))
forall(g in GENS, h in HOURS|h>2) unit_hot_started(g,h)  >= unit_started(g,h)-(1-unit_on(g,h-2))
forall(g in GENS, h in HOURS)     unit_cold_started(g,h) >= unit_started(g,h)-unit_hot_started(g,h)
```

8 Conclusions

This project was challenging for two primary reasons. First, the timing was short and Mosel was a new language for me. Learning the syntax and idiosyncrasies of a new language in three weeks is crazy. Second, while I have done this sort of optimization in the past, it has been quite some time. I had forgotten the clever ways to write constraints so that the minimization function ends up taking care of the solution for me. Bits of my MS in Math began to come back to me near the end and, after some assistance from Yushi on the Minimum Times, I was able to write the rest of the constraints on my own.

8.1 Testing

I write software for a living. If given ample time, I would write exhaustive unit tests for each constraint added. As it stands, a "concise" report was requested and we were not given very much time. In my opinion, the phrases "concise" and "thoroughly test" do not go together. All there was room for was some light testing.

I have tested my program far more than I have reported here. In place of more thorough testing, I have described the formulation in the very extensive appendices in hopes that it demonstrates my clear understanding of the subject matter.

8.2 Working With Others

I devoted a considerable amount of time to helping others learn the Mosel language and understand how to write the constraints. Given many other students' unfamiliarity with writing code, I think many of them may have adopted my coding style. I avoided giving my code out but I did write some code on whiteboards while explaining concepts. I would not be surprised if my code style works its way into their reports.

References

[1] FICO. *FICO Xpress Optimization Suite: Getting Started with Xpress*. June 2, 2009. URL: http://www.fico.com/en/node/8140?file=5136.

[2] FICO. *FICO Xpress Optimization Suite: Xpress–Mosel User guide*. June 3, 2009. URL: http://www.fico.com/en/wp-content/secure_upload/Xpress-Mosel-User-Guide.pdf.

[3] FICO. *Mosel Language: Quick Reference*. Jan. 27, 2010. URL: http://www.fico.com/en/node/8140?file=5194.

[4] Frederick S. Hillier. *Introduction to Operations Research*. 8th ed. McGraw-Hill Science/Engineering/Math, July 2004. ISBN: 9780072527445. URL: http://amazon.com/o/ASIN/0072527447/.

[5] Daniel S. Kirschen and Goran Strbac. *Fundamentals of Power System Economics*. 1st ed. Wiley, May 2004. ISBN: 9780470845721. URL: http://amazon.com/o/ASIN/0470845724/.

[6] Wayne L. Winston and Munirpallam Venkataramanan. *Introduction to Mathematical Programming: Operations Research, Vol. 1 (Book & CD-ROM)*. 4th. Thomson Learning, Oct. 2002. ISBN: 9780534359645. URL: http://amazon.com/o/ASIN/0534359647/.

A Appendix: Basic Economic Dispatch

A.1 Formulation: Power Minimums and Maximums

The maximum and minimum power constraints simply set the bounds for the *unit_power* decision variables. Since *unit_power* is the minimization decision variable, it has to be possible for it to be zero (note this will change once Unit On and Off Status is introduced). For this reason, the *is_semcont* constraint is used to say that *unit_power* can be 0 or greater than *PMIN*:

```
forall(g in GENS, h in HOURS) do
        unit_power(g,h) is_semcont PMIN(g)
        unit_power(g,h) <= PMAX(g)
end-do
```

A.2 Formulation: Load Generation Balance

For this section, I only need the sum of the *unit_power* of all generators for each hour to be equal to the *DEMAND*.

```
forall(h in HOURS) sum(g in GENS) unit_power(g,h) = DEMAND(h)
```

A.3 Formulation: Minimize Function

The minimization function at this stage is fairly basic, minimizing for cost for each hour. Normally, this would involve summing up across both hour and generator at the same time except that I want to look at hourly totals so first I sum those up:

```
forall(h in HOURS)
        HOURLY_POWERCOST(h):=sum(g in GENS) MARGINAL_PRICE(g)*unit_power(g,h)
```

And then I write a minimization for the *HOURLY_POWERCOST*:

```
minimize(sum(h in HOURS) HOURLY_POWERCOST(h))
```

A.4 Input File

```
        G:[4]
        H:[3]
        PMIN:[120,50,30,10]
        PMAX:[200,100,50,20]
        MARGINAL_PRICE:[20,30,40,50]
        DEMAND:[150,120,160]
```

A.5 Code

```
model "Portfolio optimization with LP"
uses "mmxprs" ! Use Xpress-Optimizer
                !DATAFILE:="../data/01.txt"
                DATAFILE:="../data/02_sample_data.txt"
        (!---CONSTANT---!)
```

```
                declarations
                        H: integer
                        G: integer
                        end-declarations
                        initializations from DATAFILE
                        H G
                        end-initializations
                        declarations
                        HOURS = 1..H
                        GENS = 1..G
                        PMIN: array(GENS) of real
                        PMAX: array(GENS) of real
                        MARGINAL_PRICE: array(GENS) of real
                        DEMAND: array(HOURS) of real
                end-declarations

                ! Read Data
                initializations from DATAFILE
                        PMIN PMAX MARGINAL_PRICE DEMAND
                end-initializations

        (!---VARIABLES---!)
                declarations
                        unit_power: array(GENS,HOURS) of mpvar
                end-declarations

        (!---CONSTRAINTS---!)
                ! power min and max
                forall(g in GENS, h in HOURS) do
                        unit_power(g,h) is_semcont PMIN(g)
                        unit_power(g,h) <= PMAX(g)
                end-do
                ! Load Generation Balance - Slide 47
                forall(h in HOURS) do
                        sum(g in GENS) unit_power(g,h) = DEMAND(h) ! Load
                end-do

        (!---OBJECTIVES---!)
                forall(h in HOURS) do
                        HOURLY_POWERCOST(h):=sum(g in GENS)(MARGINAL_PRICE(g)*unit_power(g,h))
                end-do
                minimize(sum(h in HOURS) HOURLY_POWERCOST(h))


        (!---SOLUTION PRINTING---!)
                writeln("Cost Total: $", getobjval)
                forall(h in HOURS) do
                        write("H",h,":Cost:$",getsol(HOURLY_POWERCOST(h)))
                        writeln("")
                        forall(g in GENS) do
                                write("P",g,":",getsol(unit_power(g,h)),"MW; ")
                        end-do
                writeln("")
                end-do
end-model
```

B Appendix: Dispatch with On and Off Status and Minimum Times

B.1 Formulation: On and Off Status

The *Unit On and Off Status* constraint itself is simple. We simply need a binary variable which is 1 whenever the power produced is greater than 0. This is the first use of "big M" notation:

```
forall(g in GENS, h in HOURS) unit_on(g,h)*M >= actual_power(g,h)
```

Note that there is a caveat to using these constraints. If the minimum power of a generator is 0 MW *and* that generator has a no–load–cost of $0.00, then it is possible for the program to set the generator to "on" with 0 power. This can be confusing but is avoidable if a no–load–cost is provided for each generator which is a reasonable solution.

The introduction of *Unit On and Off Status* also means that we now must multiply the *unit_on* binary decision variable with the *unit_power* decision variable to get the actual power output. Since this would constitute multiplying two decision variables by each other and that is not linear, we have to use "big M" notation in a clever way to get the actual power output which is stored in a new variable named *actual_power*.

```
forall(g in GENS, h in HOURS) do
        actual_power(g,h) <= M*unit_on(g,h)
        actual_power(g,h) >= 0
        actual_power(g,h) <= unit_power(g,h)
        actual_power(g,h) >= unit_power(g,h)-(1-unit_on(g,h))*M
end-do
```

The "big M" notation is a very clever idea. In this case, it is designed to work for the two values that *unit_on* can take: 1 and 0. This is best illustrated with the following example where $M = 1000$ and $unit\_power = 100$.

$$\begin{aligned} actual\_power &<= 0 & \quad actual\_power &<= 1000 \\ actual\_power &>= 0 & \quad actual\_power &>= 0 \\ actual\_power &<= 100 & \quad actual\_power &<= 100 \\ actual\_power &>= -900 & \quad actual\_power &>= 100 \end{aligned}$$

The left side shows the results when *unit_on* is zero while the right side shows when *unit_on* is one. When $unit\_on = 0$, the first two constraints make certain $actual\_power = 0$. When $unit\_on = 1$, the last two constraints make certain $actual\_power = unit\_power = 100$.

B.2 Formulation: Minimum Times

The *Unit On and Off* constraint is not very interesting to test by itself. For that reason, and in the interest of conciseness, I chose to include Minimum Up Time and Minimum Down Time constraints in the same file. Also, Minimum Times constraints make particularly heavy use of the *unit_on* array.

Minimum Times were the most difficult constraints to write. I spent nearly a week struggling with them before getting assistance from Yushi. As it turns out, my problem was that I was trying to write constraints that easily worked for everything all at once. Instead, the way we do it is to use the advantage of loops to generate the necessary constraints exhaustively.

To explain this better, I will formulate the *Minimum Up Time* constraints for a single hour for a single generator. For this example, the generator has a minimum up time of 3 and we have 4 hours total. If the generator is turned on at hour 1, then we must constrain hours 2 and 3 to be on as well. This scenario is shown in 11.

0	1	2	3	4
0	1	1	1	x

Table 11: A demonstration of how the Minimum Up Time constraints are determined. The numbers on the top represent time in hours with 1 being the first our and the 0 and less being hours prior.

Writing constraints Table 11 where $u(h)$ is the on or off value at each hour h, we get:

$$u(1) - u(0) >= u(1) \rightarrow 1 - 0 >= u(1)$$
$$u(1) - u(0) >= u(2) \rightarrow 1 - 0 >= u(2)$$
$$u(1) - u(0) >= u(3) \rightarrow 1 - 0 >= u(3)$$

As shown to the right of the arrows, the *only* way for the inequalities to hold true is if the entries on the right are equal to 1.

This pattern can be continued for each possible hour where a generator turns on. Then, we need to write constraints based on the initial up–time of the generator. The three possible representations are shown in Table 11 in the second part of the table. The first row represents "on for one hour", the second row "on for two hours", and the third "on for three hours". The constraints for the first option, "on for one hour" follow:

$$u(0) <= u(1)$$
$$u(0) <= u(2)$$

It is important to note that the number of constraints to be written for "on for n hours" depend on MIN_UP and STATUS_INIT_UP. With a good use of Mosel loops statements, the following constraints were created:

```
forall(g in GENS) do ! MINIMUM UPTIME - BOUNDARY
        lim:=minlist(MIN_UP(g),H)
        forall(i in 1..lim) unit_on(g,1) - UNITS_INIT(g) <= unit_on(g,i)
        init_lim:=minlist(MIN_UP(g) - STATUS_INIT_UP(g),H)
        forall(i in 1..init_lim) UNITS_INIT(g) <= unit_on(g,i)
end-do
forall(g in GENS, h in HOURS|h>1) do ! MINIMUM UPTIME - MAIN
        lim:=minlist(h+MIN_UP(g)-1,H)
        forall(i in h..lim) unit_on(g,h) - unit_on(g,h-1) <= unit_on(g,i)
end-do
forall(g in GENS) do ! MINIMUM DOWNTIME - BOUNDARY
        lim:=minlist(MIN_DOWN(g),H)
        forall(i in 1..lim) UNITS_INIT(g) - unit_on(g,1) <= 1-unit_on(g,i)
        init_lim:=minlist(MIN_DOWN(g) - STATUS_INIT_DOWN(g),H)
        forall(i in 1..init_lim) UNITS_INIT(g) >= unit_on(g,i)
end-do
```

```
forall(g in GENS, h in HOURS|h>1) do  ! MINIMUM DOWNTIME - MAIN
        lim:=minlist(h+MIN_DOWN(g)-1,H)
        forall(i in h..lim) unit_on(g,h-1)-unit_on(g,h) <= 1-unit_on(g,i)
end-do
```

B.3 Formulation: Minimize Function

With *On and Off Status* and *Minimum Times* added to the program, the minimize function needs to be updated. Since the *unit_on* decision variables are contained within the *actual_power* decision variables, only the *HOURLY_POWERCOST* summation needs modified. The minor change is highlighted in the following code:

```
forall(h in HOURS)
        HOURLY_POWERCOST(h):=sum(g in GENS) MARGINAL_PRICE(g)*actual_power(g,h)
```

And the actual minimization function requires no changes:

```
minimize(sum(h in HOURS) HOURLY_POWERCOST(h))
```

B.4 Input File

```
        G:[4]
        H:[3]
        PMIN:[120,50,30,10]
        PMAX:[200,100,50,20]
        MARGINAL_PRICE:[20,30,40,50]
        DEMAND:[150,120,160]MIN_UP:[2,1,1,4]
        MIN_DOWN:[2,1,1,1]
        STATUS_INIT:[1,-1,-1,1]
```

B.5 Code

```
model "Portfolio optimization with LP"
        uses "mmxprs" ! Use Xpress-Optimizer
        !DATAFILE:="../data/02.txt"
        !DATAFILE:="../data/02_hwk4_2.txt"
        DATAFILE:="../data/03_sample_data.txt"
        (!---CONSTANT---!)
        declarations
                H: integer
                G: integer
        end-declarations
        initializations from DATAFILE
                H G
        end-initializations
        declarations
                HOURS = 1..H
                GENS = 1..G

                ! Basic Economic Dispatch
                PMIN: array(GENS) of real
                PMAX: array(GENS) of real
                MARGINAL_PRICE: array(GENS) of real
                DEMAND: array(HOURS) of real
                unit_power: array(GENS,HOURS) of mpvar

                ! On and Off Status and Min Up and Min Down Times
                M = 1000
                MIN_UP: array(GENS) of integer
                MIN_DOWN: array(GENS) of integer
                STATUS_INIT: array(GENS) of integer
                unit_on: array(GENS,HOURS) of mpvar
                actual_power: array(GENS, HOURS) of mpvar
        end-declarations
        initializations from DATAFILE
                PMIN PMAX MARGINAL_PRICE DEMAND
                MIN_UP MIN_DOWN STATUS_INIT
        end-initializations

        (!---INIT STATUS CALCULATIONS---!)
        forall(g in GENS) do
                if(STATUS_INIT(g)>0) then
                        STATUS_INIT_UP(g):= STATUS_INIT(g)
                        STATUS_INIT_DOWN(g):=0
                        UNITS_INIT(g):=1
                else
                        STATUS_INIT_UP(g):= 0
                        STATUS_INIT_DOWN(g):=STATUS_INIT(g)*(-1)
                        UNITS_INIT(g):=0
                end-if
        end-do

        (!---TYPE CONSTRAINTS---!)
        forall(g in GENS, h in HOURS)do
                unit_on(g,h) is_binary
        end-do

        (!---CONSTRAINTS---!)
        ! power min and max
        forall(g in GENS, h in HOURS) do
                unit_power(g,h) >= PMIN(g)
                unit_power(g,h) <= PMAX(g)
        end-do
        ! Load Generation Balance - Slide 47
        forall(h in HOURS) do
                sum(g in GENS) actual_power(g,h) = DEMAND(h) ! Load
        end-do
        ! Linearizing the product of a binary and a continuous variable
        forall(g in GENS, h in HOURS) do
                actual_power(g,h) <= M*unit_on(g,h)
                actual_power(g,h) >= 0
                actual_power(g,h) <= unit_power(g,h)
                actual_power(g,h) >= unit_power(g,h)-(1-unit_on(g,h))*M
        end-do
        ! unit_on must be 1 when power > 0
        forall(g in GENS, h in HOURS) do
                unit_on(g,h)*M >= actual_power(g,h)
        end-do
```

```
        ! Units On/Off Status
        forall(g in GENS) do ! MINIMUM UPTIME - BOUNDARY
                lim:=minlist(MIN_UP(g),H)
                forall(i in 1..lim) unit_on(g,1)-UNITS_INIT(g)<=unit_on(g,i) ! first hour constraints
                init_lim:=minlist(MIN_UP(g)-STATUS_INIT_UP(g),H)
                forall(i in 1..init_lim) UNITS_INIT(g)<=unit_on(g,i) ! initial status constraints
        end-do
        forall(g in GENS, h in HOURS|h>1) do ! MINIMUM UPTIME - MAIN
                lim:=minlist(h+MIN_UP(g)-1,H)
                forall(i in h..lim) unit_on(g,h)-unit_on(g,h-1)<=unit_on(g,i) ! rest of hours constraints
        end-do
        forall(g in GENS) do ! MINIMUM DOWNTIME - BOUNDARY
                lim:=minlist(MIN_DOWN(g),H)
                forall(i in 1..lim) UNITS_INIT(g)-unit_on(g,1)<=1-unit_on(g,i) ! first hour constraints
                init_lim:=minlist(MIN_DOWN(g)-STATUS_INIT_DOWN(g),H)
                forall(i in 1..init_lim) UNITS_INIT(g)>=unit_on(g,i) ! initial status constraints
        end-do
        forall(g in GENS, h in HOURS|h>1) do ! MINIMUM DOWNTIME - MAIN !
                lim:=minlist(h+MIN_DOWN(g)-1,H)
                forall(i in h..lim) unit_on(g,h-1)-unit_on(g,h)<=1-unit_on(g,i)
        end-do


        (!---OBJECTIVES---!)
        forall(h in HOURS) do
                HOURLY_POWERCOST(h):=sum(g in GENS)(MARGINAL_PRICE(g)*actual_power(g,h))
        end-do
        minimize(sum(h in HOURS) HOURLY_POWERCOST(h))

        (!---SOLUTION PRINTING---!)
        writeln("Cost Total: $", getobjval)
        forall(h in HOURS) do
                write("H",h,":Cost:$",getsol(HOURLY_POWERCOST(h)))
                writeln("")
                forall(g in GENS) do
                        write("P",g,":",getsol(actual_power(g,h)),"MW; ")
                end-do
                writeln("")
        end-do
        forall(h in HOURS) do
                writeln("")
                forall(g in GENS) do
                        write(getsol(unit_on(g,h)))
                end-do
        end-do
end-model
```

C Appendix: Dispatch with No Load Cost, Startup Costs, and Reserve

C.1 Formulation: No Load Cost

I actually tried to introduce No Load Cost originally in my *Basic Economic Dispatch* but without *Unit On and Off Status*, there was no simple way to easily add it in at the correct moments. With *Unit On and Off Status* implemented, *No Load Cost* is very easy. The NO_LOAD_COST array is a set of constants and I only need to add it to the HOURLY_POWERCOST calculation from Section **A.3** while multiplying it by *unit_on*.

C.2 Formulation: Startup Costs

There are two stages to *Startup Costs* in my code. The first stage is cold–start only. Since I now have a reliable *unit_on* value which tells me if the unit is on, I can easily calculate when units start up. A generator starts up when *unit_on(h)-unit_on(h-1)=1*.

```
forall(g in GENS) do unit_started(g,1) >= unit_on(g,1) - UNITS_INIT(g)
forall(g in GENS, h in HOURS|h>1) do unit_started(g,h) >= unit_on(g,h) - unit_on(g,h-1)
```

C.3 Formulation: Reserve

During class, all problems to do with *Power Reserve* used constant values. My initial implementation of power reserve in my program also used a simple array of values. After many hours attempting to do the reserve in a more sophisticated manner where I would seek to have enough reserve to cover the largest single generation value of a single generator with all other generators, I decided to take a simpler path that others have taken and go with a reserve of 20% of demand.
Since demand is constant, this makes the calculation straight forward. For each hour, the sum of the PMAX values for all generators that are on must exceed the *DEMAND+0.2*DEMAND*. Written out as constraints:

```
forall(h in HOURS)
        (sum(g in GENS) unit_on(g,h)*PMAX(g)) >= DEMAND(h)+RESERVE_PERCENT*DEMAND(h)
```

C.4 Formulation: Minimize Function

In this case, the minimize function needed only minor adjustments to include the No Load Cost and the Startup Cost since the Reserve was calculated into the power requirements.
First I calculated the hourly startup cost:

```
forall(h in HOURS)
        HOURLY_STARTUP_COST(h):=sum(g in GENS)(STARTUP_COST(g)*unit_started(g,h))
```

And then I added that into the minimization cost:

```
forall(h in HOURS) do
        HOURLY_POWERCOST(h):=sum(g in GENS)
                (NO_LOAD_COST(g)*unit_on(g,h)+MARGINAL_PRICE(g)*actual_power(g,h))+
                HOURLY_STARTUP_COST(h)
```

C.5 Input File

```
H:[3]
G:[4]
PMIN:[120,50,30,10]
PMAX:[200,100,50,20]
MARGINAL_PRICE:[20,30,40,50]
DEMAND:[150,120,160]
MIN_UP:[3,2,1]
MIN_DOWN:[3,2,1]
STATUS_INIT:[-5,3,3]
NO_LOAD_COST: [0,0,0]
STARTUP_COST: [1000,600,150]
RESERVE_PERCENT: [0.2]
```

C.6 Code

```
model "Portfolio optimization with LP"
        uses "mmxprs" ! Use Xpress-Optimizer
        !DATAFILE:="../data/03.txt"
        !DATAFILE:="../data/03_hwk4_2.txt"
        DATAFILE:="../data/03_sample_data.txt"
        (!---CONSTANT---!)
        declarations
                H: integer
                G: integer
        end-declarations
        initializations from DATAFILE
                H G
        end-initializations
        declarations
                HOURS = 1..H
                GENS = 1..G

                ! Basic Economic Dispatch
                PMIN: array(GENS) of real
                PMAX: array(GENS) of real
                MARGINAL_PRICE: array(GENS) of real
                DEMAND: array(HOURS) of real
                unit_power: array(GENS,HOURS) of mpvar

                ! On and Off Status and Min Up and Min Down Times
                M = 1000
                MIN_UP: array(GENS) of integer
                MIN_DOWN: array(GENS) of integer
                STATUS_INIT: array(GENS) of integer
                unit_on: array(GENS,HOURS) of mpvar
                actual_power: array(GENS,HOURS) of mpvar

                ! No Load Cost, Startup Cost, Reserve
                NO_LOAD_COST: array(GENS) of real
                STARTUP_COST: array(GENS) of real
                RESERVE_PERCENT: real
                unit_started: array(GENS,HOURS) of mpvar
        end-declarations
```

```
initializations from DATAFILE
        PMIN PMAX MARGINAL_PRICE DEMAND
        MIN_UP MIN_DOWN STATUS_INIT
        NO_LOAD_COST STARTUP_COST RESERVE_PERCENT
end-initializations

(!---INIT STATUS CALCULATIONS---!)
forall(g in GENS) do
        if(STATUS_INIT(g)>0) then
                STATUS_INIT_UP(g):= STATUS_INIT(g)
                STATUS_INIT_DOWN(g):=0
                UNITS_INIT(g):=1
        else
                STATUS_INIT_UP(g):= 0
                STATUS_INIT_DOWN(g):=STATUS_INIT(g)*(-1)
                UNITS_INIT(g):=0
        end-if
end-do

(!---TYPE CONSTRAINTS---!)
forall(g in GENS, h in HOURS)do
        unit_on(g,h) is_binary
        unit_started(g,h) is_binary
end-do

(!---CONSTRAINTS---!)
! power min and max
forall(g in GENS, h in HOURS) do
        unit_power(g,h) >= PMIN(g)
        unit_power(g,h) <= PMAX(g)
end-do
! Load Generation Balance - Slide 47
forall(h in HOURS) do
        sum(g in GENS) actual_power(g,h) = DEMAND(h) ! Load
end-do
! Linearizing the product of a binary and a continuous variable
forall(g in GENS, h in HOURS) do
        actual_power(g,h) <= M*unit_on(g,h)
        actual_power(g,h) >= 0
        actual_power(g,h) <= unit_power(g,h)
        actual_power(g,h) >= unit_power(g,h)-(1-unit_on(g,h))*M
end-do
! unit_on must be 1 when power > 0
forall(g in GENS, h in HOURS) do
        unit_on(g,h)*M >= actual_power(g,h)
end-do
! Units On/Off Status
forall(g in GENS) do ! MINIMUM UPTIME - BOUNDARY
        lim:=minlist(MIN_UP(g),H)
        forall(i in 1..lim) unit_on(g,1)-UNITS_INIT(g)<=unit_on(g,i) ! first hour constraints
        init_lim:=minlist(MIN_UP(g)-STATUS_INIT_UP(g),H)
        forall(i in 1..init_lim) UNITS_INIT(g)<=unit_on(g,i) ! initial status constraints
end-do
forall(g in GENS, h in HOURS|h>1) do ! MINIMUM UPTIME - MAIN
        lim:=minlist(h+MIN_UP(g)-1,H)
```

```
		forall(i in h..lim) unit_on(g,h)-unit_on(g,h-1)<=unit_on(g,i) ! rest of hours constraints
	end-do
	forall(g in GENS) do ! MINIMUM DOWNTIME - BOUNDARY
		lim:=minlist(MIN_DOWN(g),H)
		forall(i in 1..lim) UNITS_INIT(g)-unit_on(g,1)<=1-unit_on(g,i) ! first hour constraints
		init_lim:=minlist(MIN_DOWN(g)-STATUS_INIT_DOWN(g),H)
		forall(i in 1..init_lim) UNITS_INIT(g)>=unit_on(g,i) ! initial status constraints
	end-do
	forall(g in GENS, h in HOURS|h>1) do ! MINIMUM DOWNTIME - MAIN !
		lim:=minlist(h+MIN_DOWN(g)-1,H)
		forall(i in h..lim) unit_on(g,h-1)-unit_on(g,h)<=1-unit_on(g,i)
	end-do
	! Startup Status !
	forall(g in GENS) unit_started(g,1)>=unit_on(g,1)-UNITS_INIT(g)
	forall(g in GENS, h in HOURS|h>1) unit_started(g,h) >= unit_on(g,h)-unit_on(g,h-1)
	! Startup Cost Total
	forall(h in HOURS)
		HOURLY_STARTUP_COST(h):=sum(g in GENS)(STARTUP_COST(g)*unit_started(g,h))
	! Reserve: Percentage of Demand
	forall(h in HOURS)
		(sum(g in GENS) unit_on(g,h)*PMAX(g)) >= DEMAND(h)+RESERVE_PERCENT*DEMAND(h)

	(!---OBJECTIVES---!)
	forall(h in HOURS) do
		HOURLY_POWERCOST(h):=sum(g in GENS)
			(NO_LOAD_COST(g)*unit_on(g,h)+MARGINAL_PRICE(g)*actual_power(g,h))+
			HOURLY_STARTUP_COST(h)
	end-do
	minimize(sum(h in HOURS) HOURLY_POWERCOST(h))

	(!---SOLUTION PRINTING---!)
	writeln("Cost Total: $", getobjval)
	forall(h in HOURS) do
		write("H",h,":Cost:$",getsol(HOURLY_POWERCOST(h)))
		writeln("")
		forall(g in GENS) do
			write("P",g,":",getsol(actual_power(g,h)),"MW; ")
		end-do
	writeln("")
	end-do
	writeln("")
	writeln("unit_on")
	forall(g in GENS) write(getsol(UNITS_INIT(g)))
	write("(UNITS_INIT)")
	writeln("")
	writeln("---")
	forall(h in HOURS) do
		forall(g in GENS) do
			write(getsol(unit_on(g,h)))
		end-do
		writeln("")
	end-do
	writeln("")
	writeln("unit_started")
	forall(h in HOURS) do
```

```
                forall(g in GENS) do
                        write(getsol(unit_started(g,h)))
                end-do
                writeln("")
        end-do
end-model
```

D Appendix: Dispatch with Three–Segment Cost Curves

D.1 Formulation: Three–Segment Cost Curves

Three–segment cost curves are done by splitting the power generated by a single generator into three sections each with their own marginal cost. The cost is then calculated across all three "generators".

Explaining this with an example, if I have a single generator that can generate between 200MW and 500MW which has a no–load cost of $400 and I want to use a three–segment cost curve, it would be divided as shown in Table 12.

Generator	Power	Cost
0	200 MW	$400
1	0 to 100 MW	$15/MWh
2	0 to 100 MW	$20/MWh
3	0 to 100 MW	$25/MWh

Table 12: An example of how a 200MW to 500MW generator would have its generation and cost divided into three segments.

The *unit_power* value which would have normally been between 200MW and 500MW is instead calculated as the the sum of 200MW + the value of each of three 100MW "generators":

The following code shows the calculation of the maximum power values and setting the range of each generator at those limits.

```
forall(g in GENS) PMAX_THREE_SEG(g):=(PMAX(g) - PMIN(g))/P
forall(p in PRICES, g in GENS, h in HOURS) do
        split_power(p,g,h) >= 0
        split_power(p,g,h) <= PMAX_THREE_SEG(g)
end-do
```

The final step is to write *unit_power* as the sum across the three small generators. Since *actual_power* is calculated using *unit_power* and it works with the *unit_on* binary, there is no need to worry about having *unit_power* never be zero.

```
forall(g in GENS, h in HOURS)
        unit_power(g,h) = PMIN(g) + sum(p in PRICES)(split_power(p,g,h))
```

Since this code effects the calculation of *unit_power*, it was added near the top of the constraints in the program.

D.2 Formulation: Minimize Function

Since the cost is separate, the minimization functions need minor adjustments to calculate price on a per–segment basis.

```
forall(g in GENS, h in HOURS)
        HOURLY_GEN_COST(g,h):=NO_LOAD_COST(g)*unit_on(g,h)+
                PMIN(g)*PRICE(1,g)*unit_on(g,h)+
                (sum(p in PRICES)PRICE(p,g)*split_power(p,g,h))
forall(h in HOURS)
        HOURLY_POWERCOST(h):=sum(g in GENS)(HOURLY_GEN_COST(g,h))+HOURLY_STARTUP_COST(h)
```

D.3 Input File

```
H:[6]
G:[4]
P:[3]
PMIN:[25,50,60,0]
PMAX:[100,150,200,250]
MARGINAL_PRICE:[15.45,17.3467,26.9,36.9]
DEMAND:[250,200,390,250,130,300]
MIN_UP:[1,1,3,2]
MIN_DOWN:[1,2,2,1]
STATUS_INIT:[2,-3,-5,-7]
NO_LOAD_COST:[0,0,0,0]
STARTUP_COST:[500,700,800,900]
RESERVE_PERCENT:[0.2]
PRICE:[15,17,26,36,
          20,25,30,40,
       30,35,40,50]
```

D.4 Code

```
model "Portfolio optimization with LP"
        uses "mmxprs" ! Use Xpress-Optimizer
        !DATAFILE:="../data/04.txt"
        !DATAFILE:="../data/04_hwk4_2.txt"
        !DATAFILE:="../data/04_hwk8_3.txt"
        DATAFILE:="../data/04_sample_data.txt"
        (!---CONSTANT---!)
        declarations
                H: integer
                G: integer
                P: integer
        end-declarations
        initializations from DATAFILE
                H G P
        end-initializations
        declarations
                HOURS = 1..H
                GENS = 1..G

                ! Basic Economic Dispatch
                PMIN: array(GENS) of real
                PMAX: array(GENS) of real
                MARGINAL_PRICE: array(GENS) of real
                DEMAND: array(HOURS) of real
                unit_power: array(GENS,HOURS) of mpvar

                ! On and Off Status and Min Up and Min Down Times
                M = 1000
                MIN_UP: array(GENS) of integer
                MIN_DOWN: array(GENS) of integer
                STATUS_INIT: array(GENS) of integer
                unit_on: array(GENS,HOURS) of mpvar
                actual_power: array(GENS,HOURS) of mpvar
                ! No Load Cost, Startup Cost, Reserve
                NO_LOAD_COST: array(GENS) of real
                STARTUP_COST: array(GENS) of real
                RESERVE_PERCENT: real
                unit_started: array(GENS,HOURS) of mpvar

                ! Three Segment Cost Curves
                PRICES = 1..P
                PRICE: array(PRICES,GENS) of real
                split_power: array(PRICES,GENS,HOURS) of mpvar
        end-declarations
        initializations from DATAFILE
                PMIN PMAX MARGINAL_PRICE DEMAND
                MIN_UP MIN_DOWN STATUS_INIT
                NO_LOAD_COST STARTUP_COST RESERVE_PERCENT
                PRICE
        end-initializations

        (!---INIT STATUS CALCULATIONS---!)
        forall(g in GENS) do
                if(STATUS_INIT(g)>0) then
                        STATUS_INIT_UP(g):= STATUS_INIT(g)
                        STATUS_INIT_DOWN(g):=0
                        UNITS_INIT(g):=1
                else
                        STATUS_INIT_UP(g):= 0
                        STATUS_INIT_DOWN(g):=STATUS_INIT(g)*(-1)
                        UNITS_INIT(g):=0
                end-if
        end-do
        forall(g in GENS) MARGINAL_PRICE(g) := PRICE(1,g)

        (!---TYPE CONSTRAINTS---!)
        forall(g in GENS, h in HOURS)do
                unit_on(g,h) is_binary
                unit_started(g,h) is_binary
        end-do

        (!---CONSTRAINTS---!)
        ! power min and max
        forall(g in GENS, h in HOURS) do
                unit_power(g,h) >= PMIN(g)
                unit_power(g,h) <= PMAX(g)
        end-do
        ! Load Generation Balance - Slide 47
        forall(h in HOURS) do
                sum(g in GENS) actual_power(g,h) = DEMAND(h) ! Load
        end-do
        ! Linearizing the product of a binary and a continuous variable
        forall(g in GENS, h in HOURS) do
                actual_power(g,h) <= M*unit_on(g,h)
                actual_power(g,h) >= 0
                actual_power(g,h) <= unit_power(g,h)
                actual_power(g,h) >= unit_power(g,h)-(1-unit_on(g,h))*M
```

```
end-do
! unit_on must be 1 when power > 0
forall(g in GENS, h in HOURS) do
        unit_on(g,h)*M >= actual_power(g,h)
end-do
! Units On/Off Status
forall(g in GENS) do ! MINIMUM UPTIME - BOUNDARY
        lim:=minlist(MIN_UP(g),H)
        forall(i in 1..lim) unit_on(g,1)-UNITS_INIT(g)<=unit_on(g,i) ! first hour constraints
        init_lim:=minlist(MIN_UP(g)-STATUS_INIT_UP(g),H)
        forall(i in 1..init_lim) UNITS_INIT(g)<=unit_on(g,i) ! initial status constraints
end-do
forall(g in GENS, h in HOURS|h>1) do ! MINIMUM UPTIME - MAIN
        lim:=minlist(h+MIN_UP(g)-1,H)
        forall(i in h..lim) unit_on(g,h)-unit_on(g,h-1)<=unit_on(g,i) ! rest of hours constraints
end-do
forall(g in GENS) do ! MINIMUM DOWNTIME - BOUNDARY
        lim:=minlist(MIN_DOWN(g),H)
        forall(i in 1..lim) UNITS_INIT(g)-unit_on(g,1)<=1-unit_on(g,i) ! first hour constraints
        init_lim:=minlist(MIN_DOWN(g)-STATUS_INIT_DOWN(g),H)
        forall(i in 1..init_lim) UNITS_INIT(g)>=unit_on(g,i) ! initial status constraints
end-do
forall(g in GENS, h in HOURS|h>1) do ! MINIMUM DOWNTIME - MAIN !
        lim:=minlist(h+MIN_DOWN(g)-1,H)
        forall(i in h..lim) unit_on(g,h-1)-unit_on(g,h)<=1-unit_on(g,i)
end-do
! Startup Status !
forall(g in GENS) unit_started(g,1)>=unit_on(g,1)-UNITS_INIT(g)
forall(g in GENS, h in HOURS|h>1) unit_started(g,h) >= unit_on(g,h)-unit_on(g,h-1)
! Startup Cost Total
forall(h in HOURS)
        HOURLY_STARTUP_COST(h):=sum(g in GENS)(STARTUP_COST(g)*unit_started(g,h))
! Reserve: Percentage of Demand
forall(h in HOURS)
        (sum(g in GENS) unit_on(g,h)*PMAX(g)) >= DEMAND(h)+RESERVE_PERCENT*DEMAND(h)
! three-segment cost curves
forall(g in GENS) PMAX_THREE_SEG(g):=(PMAX(g)-PMIN(g))/P
forall(p in PRICES, g in GENS, h in HOURS) do
        split_power(p,g,h) >= 0
        split_power(p,g,h) <= PMAX_THREE_SEG(g)
end-do
forall(g in GENS, h in HOURS)
        unit_power(g,h) = PMIN(g)+sum(p in PRICES)(split_power(p,g,h))

(!---OBJECTIVES---!)
! Three Segment Price
forall(g in GENS, h in HOURS)
        HOURLY_GEN_COST(g,h):=NO_LOAD_COST(g)*unit_on(g,h)+
                MARGINAL_PRICE(g)*PMIN(g)*unit_on(g,h)+
                sum(p in PRICES)(PRICE(p,g)*split_power(p,g,h))
forall(h in HOURS) do
        HOURLY_POWERCOST(h):=sum(g in GENS)(HOURLY_GEN_COST(g,h))+HOURLY_STARTUP_COST(h)
end-do
minimize(sum(h in HOURS) HOURLY_POWERCOST(h))
```

```
        (!---SOLUTION PRINTING---!)
        writeln("Cost Total: $", getobjval)
        forall(h in HOURS) do
                write("H",h,":Cost:$",getsol(HOURLY_POWERCOST(h)))
                writeln("")
                forall(g in GENS) do
                        write("P",g,":",getsol(actual_power(g,h)),"MW; ")
                end-do
                writeln("")
        end-do
        writeln("")
        writeln("unit_on")
        forall(g in GENS) write(getsol(UNITS_INIT(g)))
        write("(UNITS_INIT)")
        writeln("")
        writeln("---")
        forall(h in HOURS) do
                forall(g in GENS) do
                        write(getsol(unit_on(g,h)))
                end-do
                writeln("")
        end-do
end-model
```