Assignment - 2

Electricity Provisioning based on Demand Prediction

AMS 559

Date - December 2nd, 2018

Group 1

Rohan Vaish (111447435)
Jatin Sood (112079000)
Harsha Chandwani (111481387)
(Team Leader)

Task 1

We have used cvxpy python library to solve the convex equation for optimized provisioning.
The offline solutions have been computed for 10 homes for the first week in November, 2015 (Nov 1, 2015 to Nov 7, 2015). We have used the actual energy demand (in kW) for each 15-minute interval that covers this week. *y(t)* represents the energy demand at time slot t (given in Homework 1) and *x(t)* represents the energy provisioned at time slot t using the offline provisioning algorithms.

a. Offline Static Provisioning

We use the cost equation given in the assignment, except the switching cost part, to find the optimal provisioning values.

```
equation = p*x + a*(cp.maximum(0,cp.exp(y-x)))
objective = cp.Minimize(cp.sum(equation))
constraints = [0 <= x]
prob = cp.Problem(objective, constraints)

# The optimal objective value is returned by `prob.solve()`.
result = prob.solve()
print(result)
# The optimal value for x is stored in `x.value`.
print(x.value)
```

b. Offline Dynamic Provisioning

In Dynamic Provisioning the switching cost is also taken into consideration. Here, x values will try to be as close as possible to y values.

```
#Offline Dynamic Provisioning
p = 0.1
a = 1
x = cp.Variable(672)
sum = 0

for i in range(1,672):
    sum += (cp.multiply(p,x[i]) + cp.maximum(0, y[i]-x[i]) + cp.abs(x[i] - x[i-1]))

objective = (cp.Minimize(sum))
constraints = [0 <= x]
prob = cp.Problem(objective, constraints)

# The optimal objective value is returned by `prob.solve()`.
result = prob.solve()
print(result)
# The optimal value for x is stored in `x.value`.
print(x.value)
```

The shown method tries to find the optimal values of provisioning, i.e *x,* such that sum of cost of the function at all values of x is minimized.

Below is the table of observed costs for each home for both offline static and dynamic solutions:

Users	Offline Static cost ($)	Offline Dynamic cost ($)
Home 1	293.07690	196.90778
Home 2	296.61887	225.17784
Home 3	292.52307	195.38877
Home 4	273.28092	165.32679
Home 5	281.04448	203.640186
Home 6	287.59182	243.55150
Home 7	319.20115	244.295137
Home 8	303.87731	248.77355
Home 9	283.69274	181.86941
Home 10	280.11231	169.127969

Table showing the optimized costs for all the 10 users

***Observation*:** Here we can see that the Dynamic Offline Provisioning gives better cost values than Static Offline Provisioning for all users. The cost for Home 4 is minimum of all the houses with both the algorithms.

Below are the provisioning graphs for all the 10 houses that show the energy values provisioned (in KW) using the offline Dynamic method against the true energy demand. The observations from these plots are mentioned after the graphs.

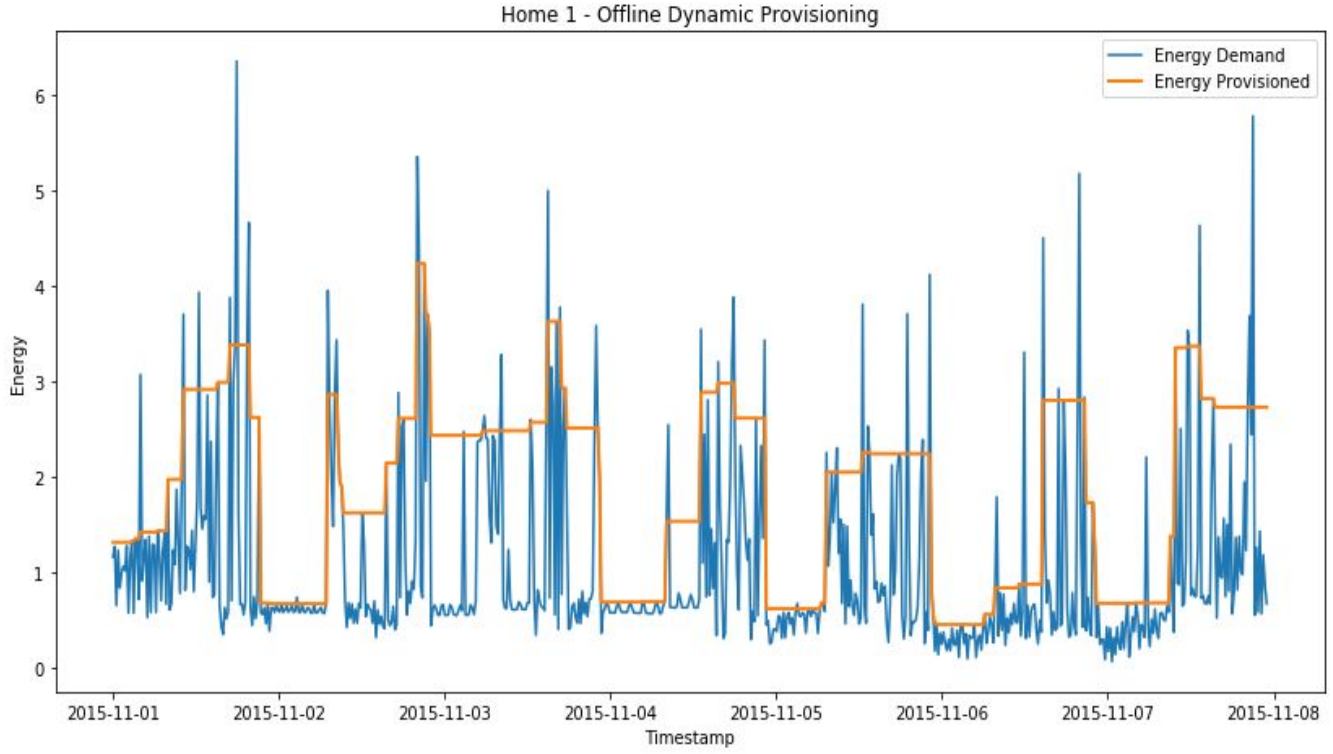

Figure 1: Home 1-Offline Dynamic Provisioning

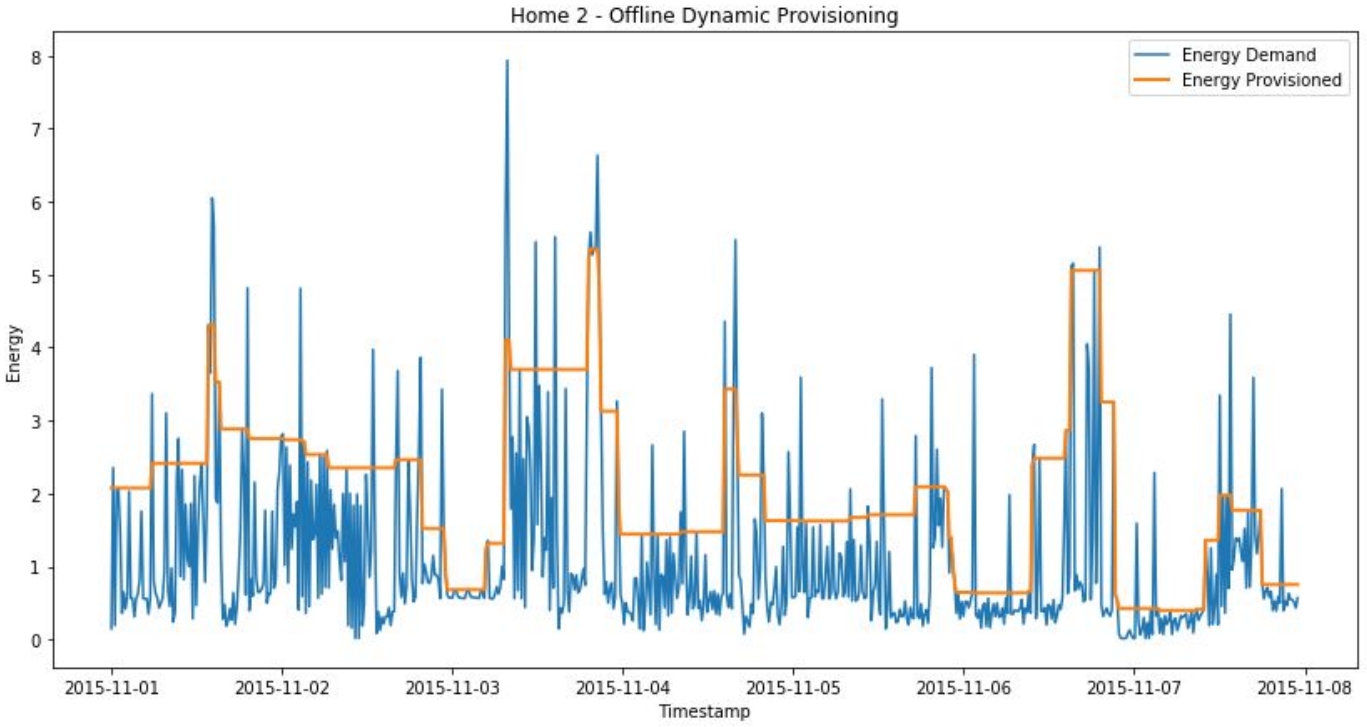

Figure 2: Home 2-Offline Dynamic Provisioning

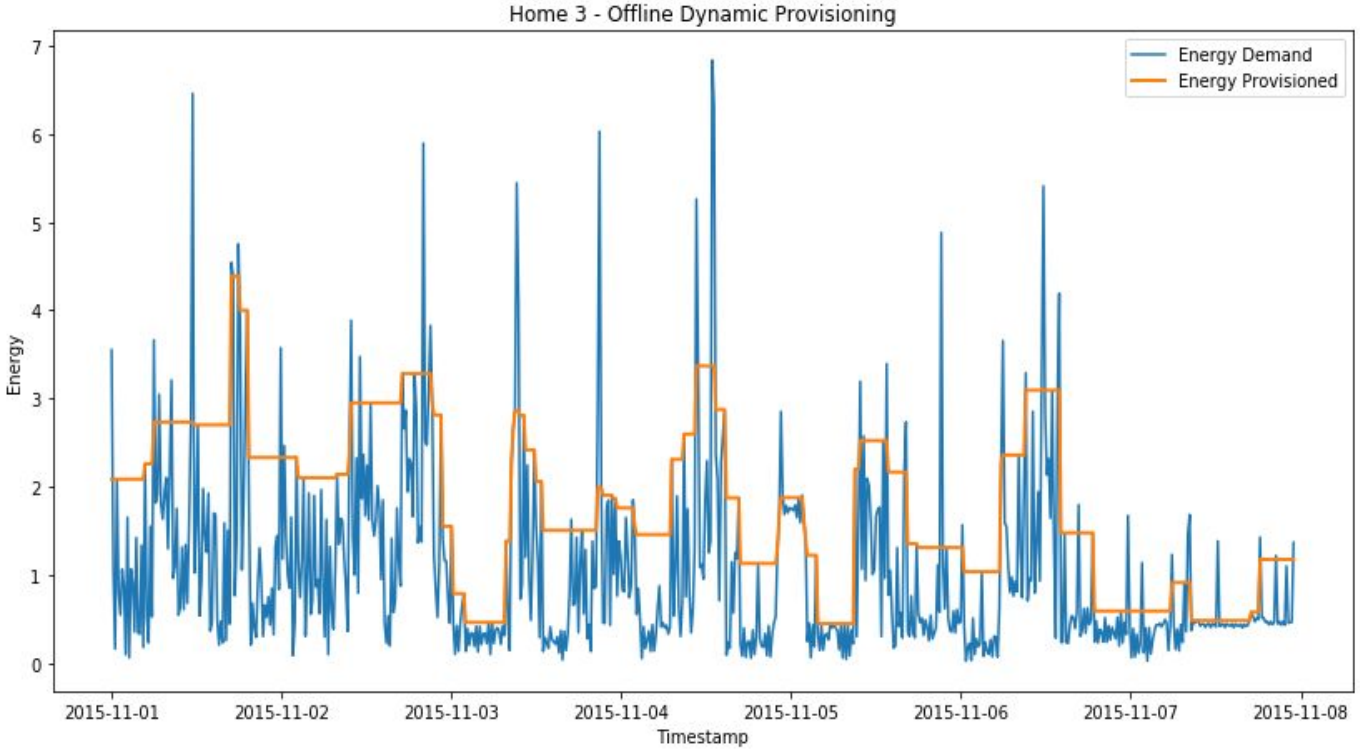

Figure 3 : Home 3 -Offline Dynamic Provisioning

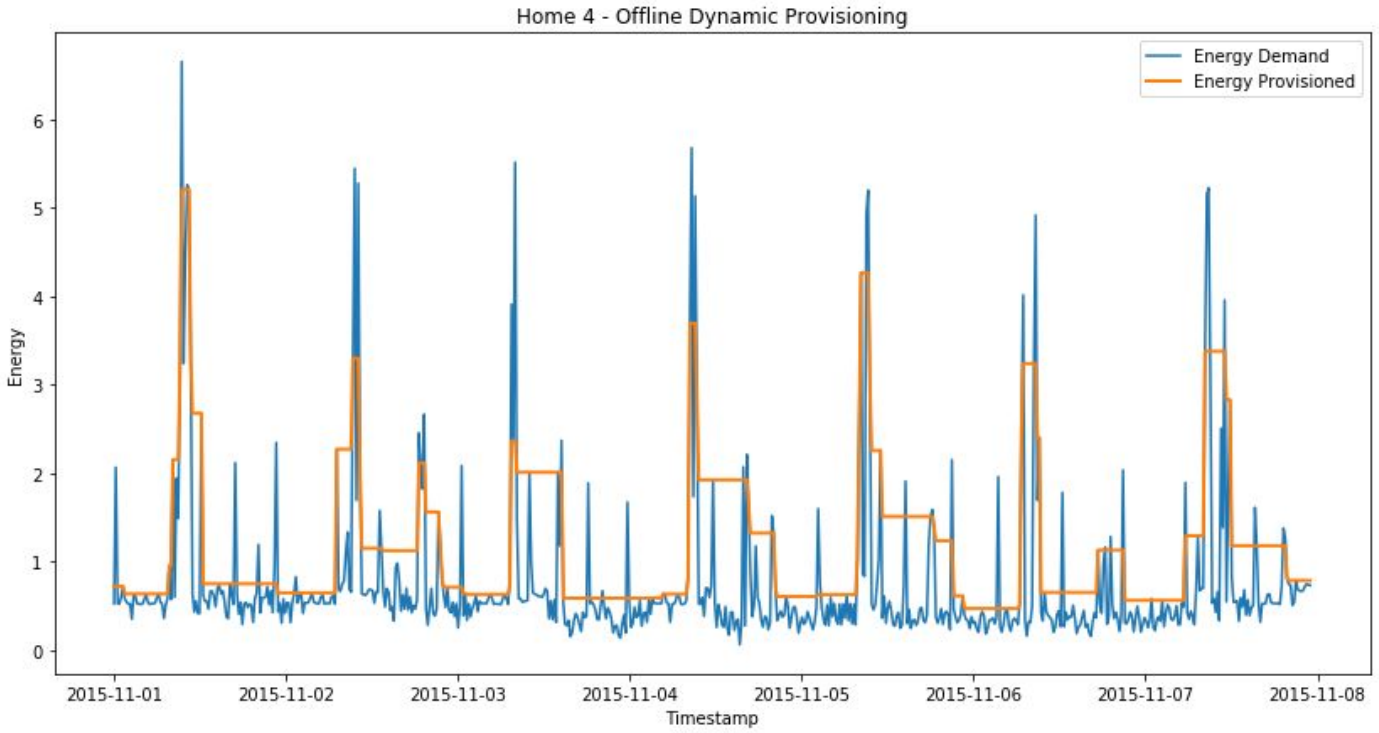

Figure 4 : Home 4 -Offline Dynamic Provisioning

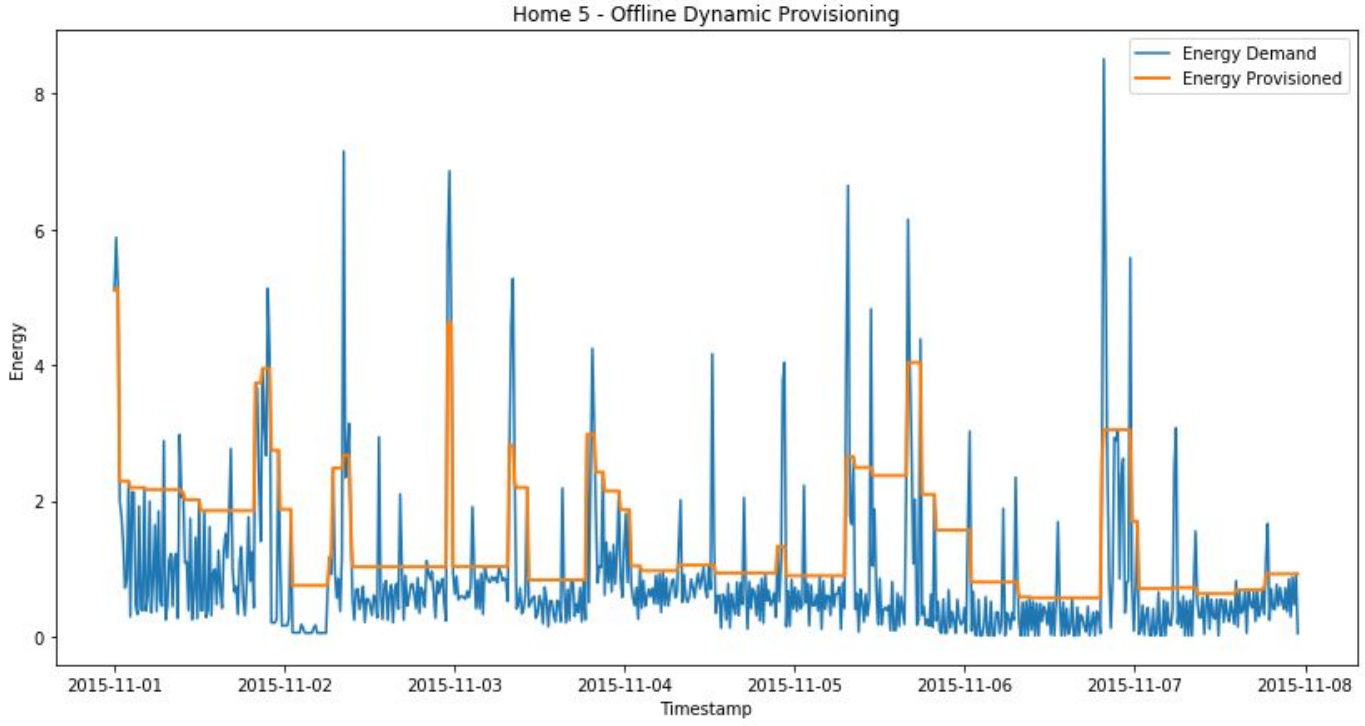

Figure 5 : Home 5 -Offline Dynamic Provisioning

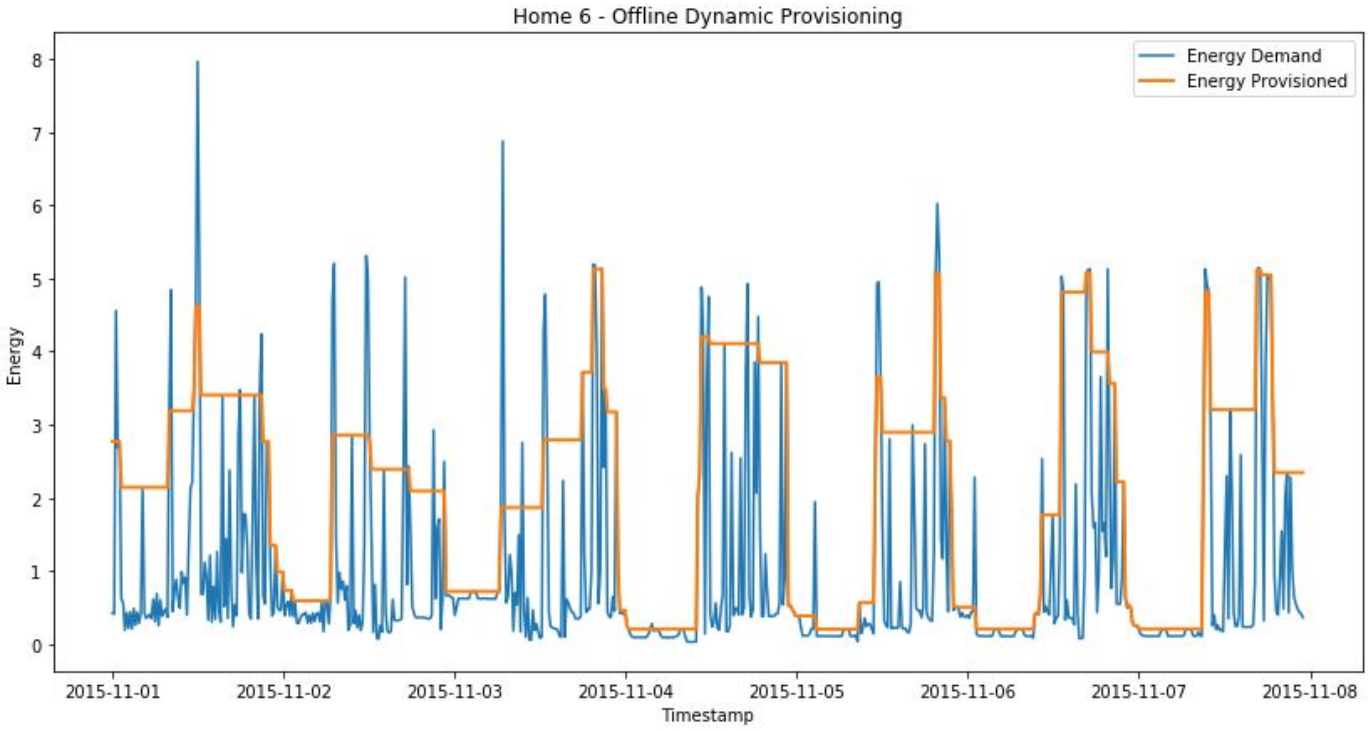

Figure 6 : Home 6 -Offline Dynamic Provisioning

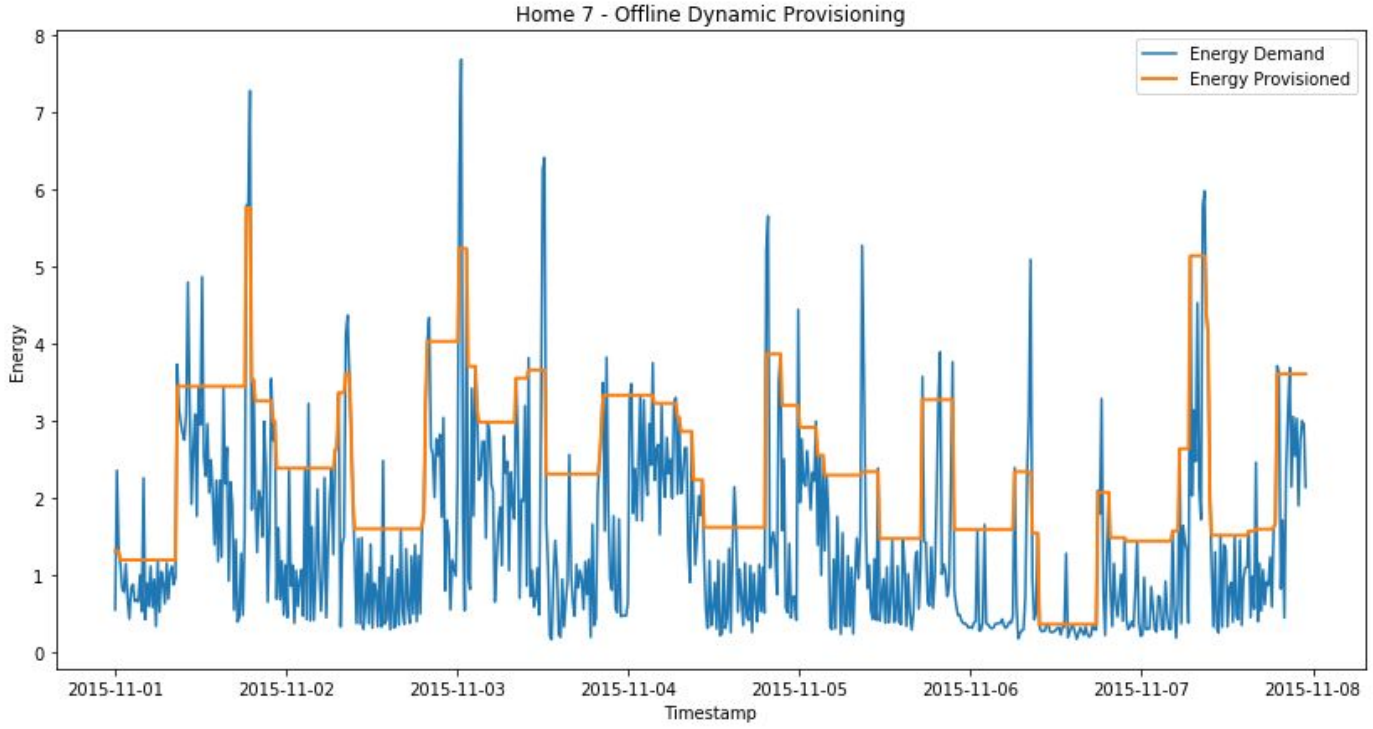

Figure 7 : Home 7 -Offline Dynamic Provisioning

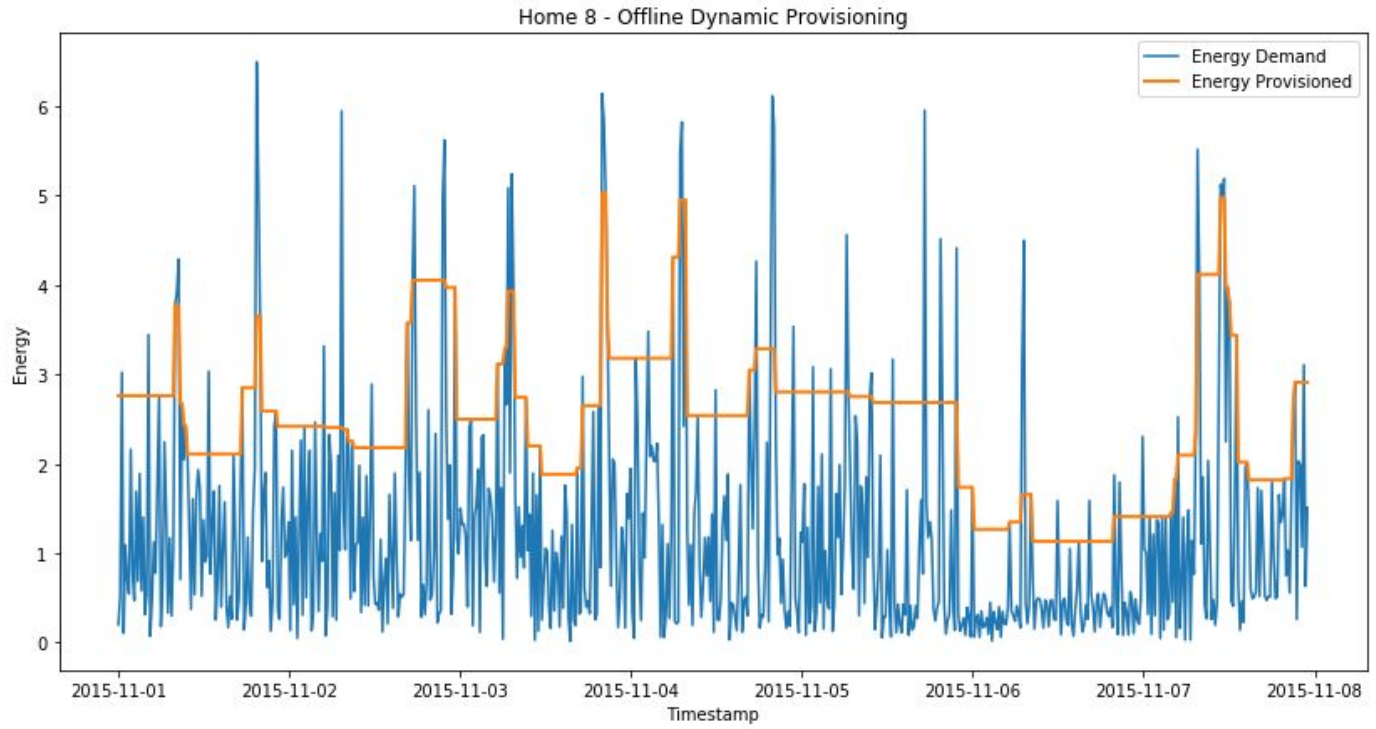

Figure 8 : Home 8 -Offline Dynamic Provisioning

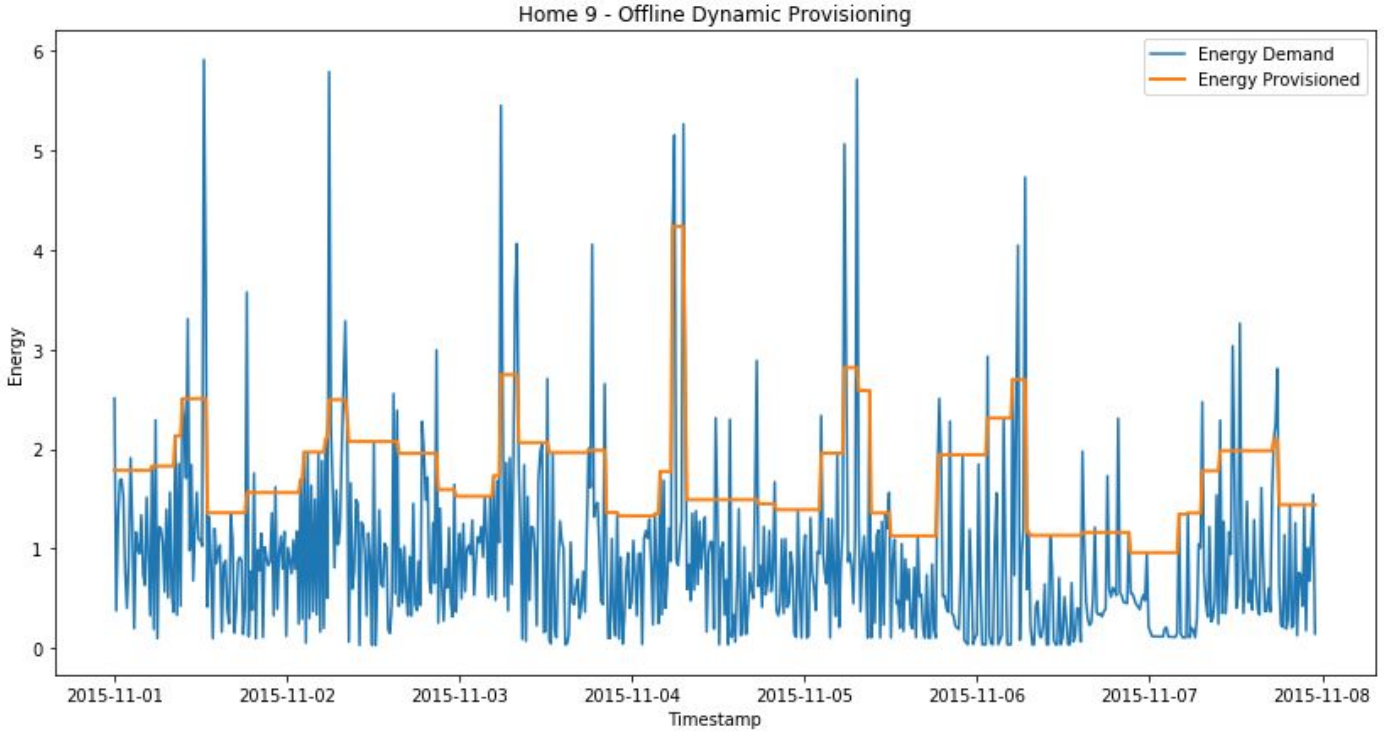

Figure 9 : Home 9 -Offline Dynamic Provisioning

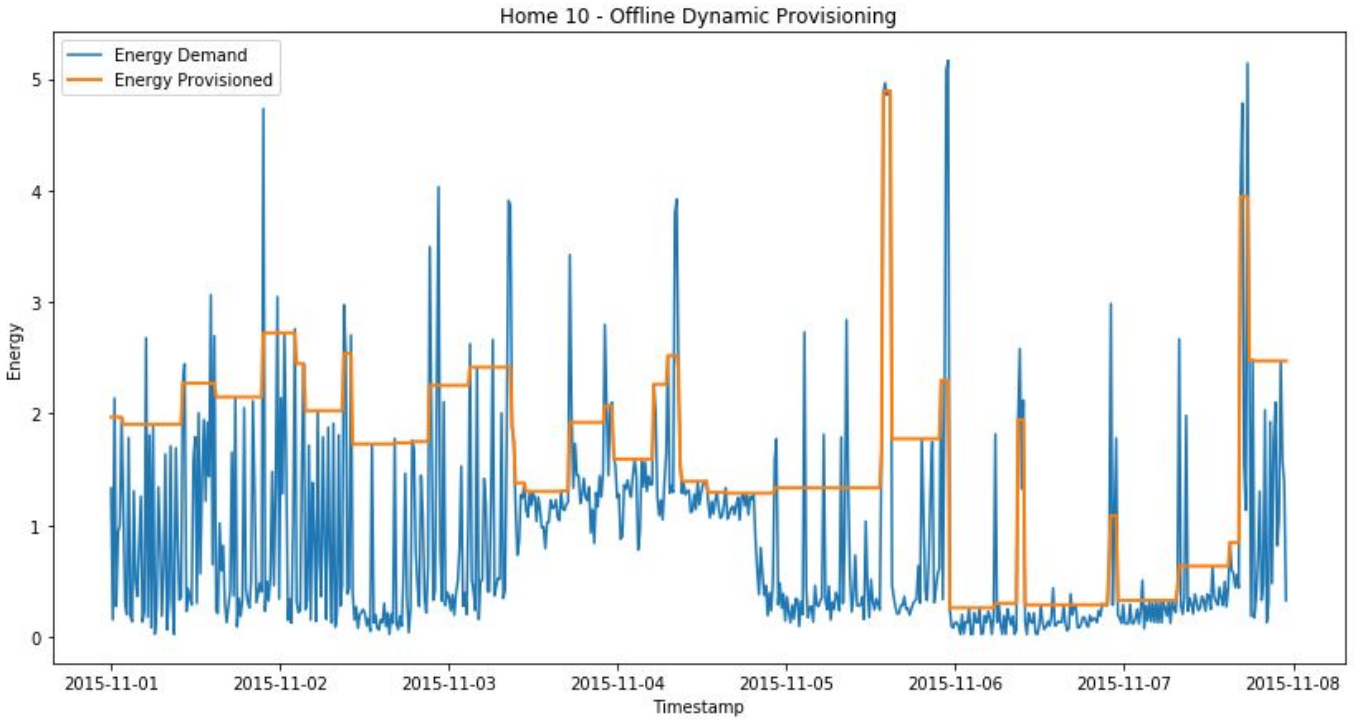

Figure 10 : Home 10 -Offline Dynamic Provisioning

Observations from offline provisioning:

1. For all the homes, we can observe that the dynamic provisioning is consistent in behavior i.e. there are no abrupt peaks in the graph.
2. The X values are trying to be as close as possible to the Y values, and this is best seen in the graph of Home 4.
3. The static provisioning does not try to match the Y values as it does not consider the switching cost. If we plot the graph for static algorithm, we can see that the X values do not try to match with Y.

Task 2

1) Prediction Plots for the Online Algorithms.

a. Online Gradient Descent

Online Gradient Descent uses the previous provisioning value, learning rate and gradient of the cost function at the previous provisioning value to compute the current provisioning value.

The *y* values are the actual energy demand values of houses 1, 2 and 3. We use the switching cost in the calculation of derivative of function at a particular provisioning value.

After multiple trials, we found that the value of learning rate, i.e η = 0.055 produces most optimum results for all the houses. Hence, we choose to show only the best results here.

Below is the representation for Home 1, 2 and 3 predictions for the Online Gradient Descent Algorithm:

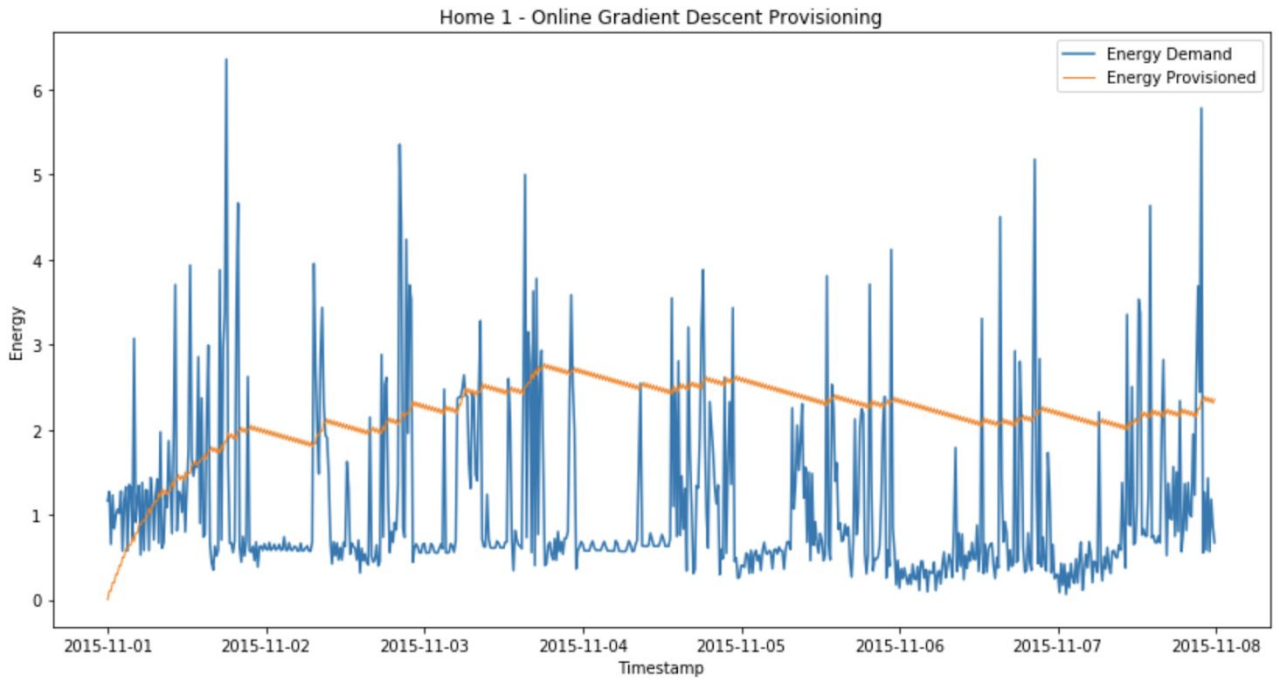

Figure 11: Home 1 - Online Gradient Descent Provisioning
Optimized cost : $285.29625

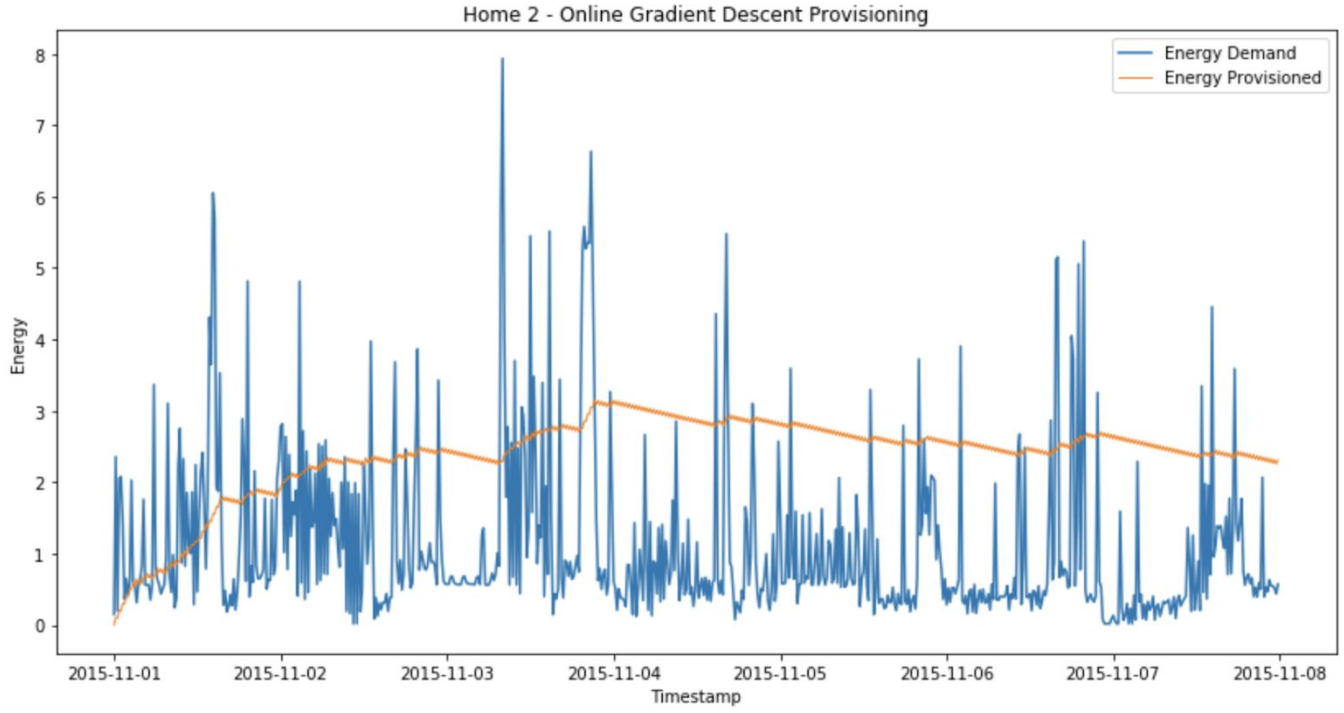

Figure 12: Home 2 - Online Gradient Descent Provisioning
Optimized cost : $328.16781

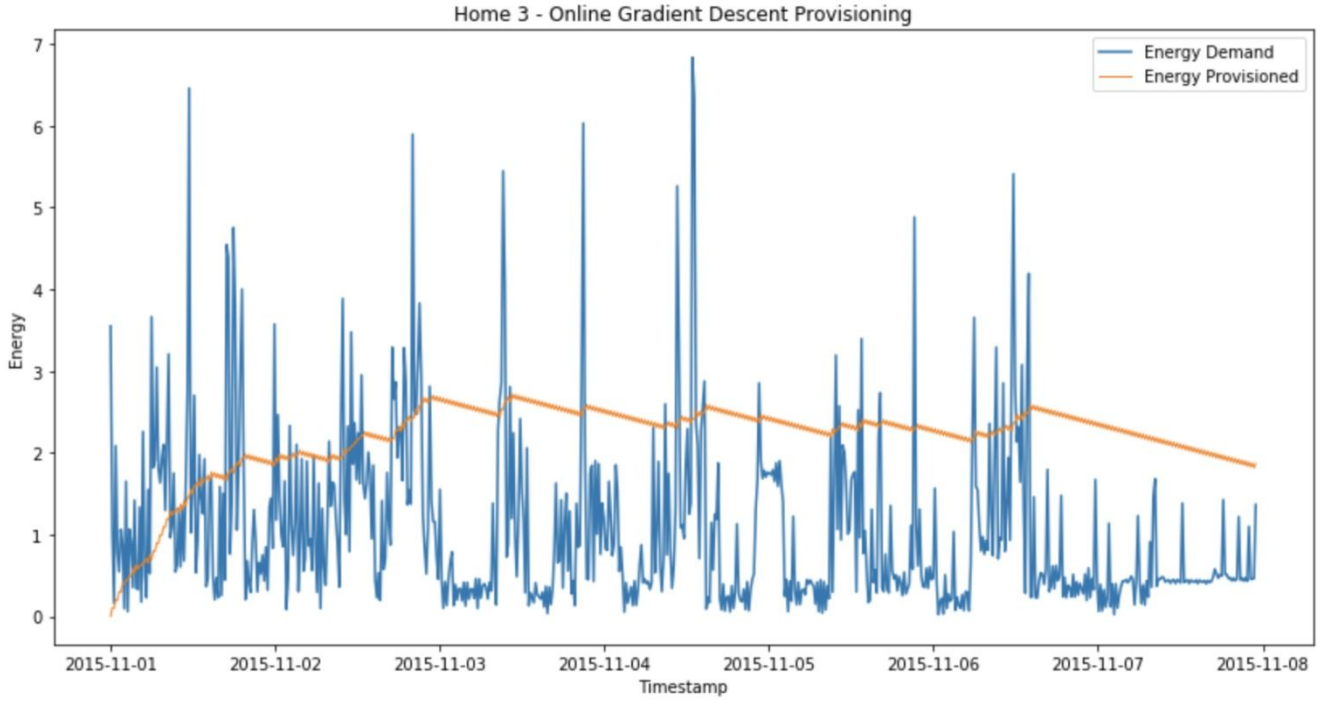

Figure 13: Home 3 - Online Gradient Descent Provisioning
Optimized cost : $292.54329

Observation from OGD: The Online Gradient Descent doesn't perform as good as the offline Dynamic provisioning solution. The optimized cost obtained using the OGD algorithm for the three houses is greater when compared to the offline Dynamic algorithm.

b. Receding Horizon Control

Here, for each calculation of the provisioning value *x*, the sum of the given cost function is minimized over the next *w* windows and the optimized value of the first window is assigned to the current index *x*. This is done for each 15 minute interval of the given week. We use the switching cost for calculating the optimized provisioning values.

We use the energy prediction values over the given week from the assignment 1 algorithms: Arima time series prediction and Adaboost regression. After multiple trials with varying window size, we found that window size of 12 is optimal for ARIMA predictions and window size of 5 is optimal for adaboost.

c. Commitment Horizon Control

Here, for each calculation of the provisioning value *x*, the sum of the given cost function is minimized over the next *w* windows and the average of the first *v* windows is committed to the current index *x*. Similar to RHC, this is done for each 15 minute interval of the given week and we use the switching cost for calculating the optimized provisioning values. Again, similar to RHC, we compute the optimized provisioning values for both ARIMA and adaboost predictions.

After multiple trials keeping window size of 12 for CHC with arima predictions and window size of 5 for CHC with adaboost predictions, we found that commitment window size (*v*) of 2 is optimal.

Below is the representation of Home 1 provisioning values from the RHC and CHC algorithms using the prediction values obtained from the ARIMA and AdaBoost models. The RHC and CHC results overlap very closely for majority of the timeline, hence orange line (RHC) is not visible.

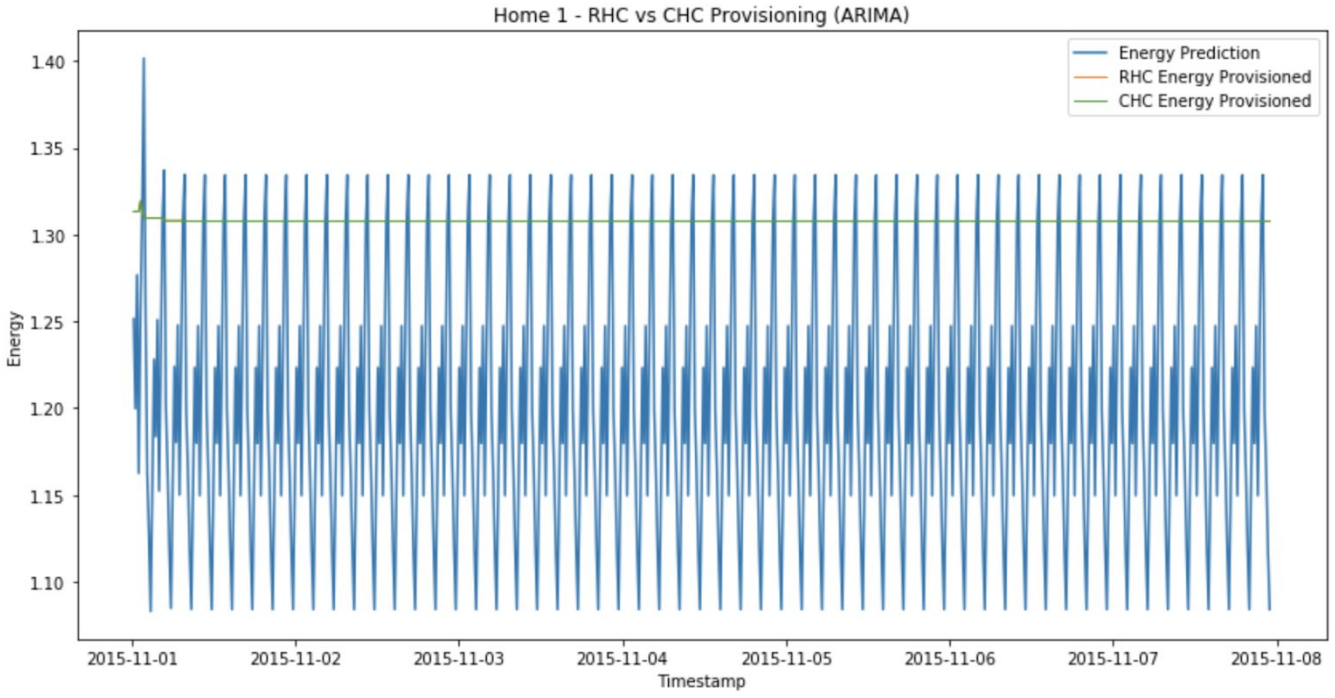

Figure 14: Home 1 - RHC (w = 12) vs CHC (w = 12, v = 2) using ARIMA predictions
RHC Cost $90.64281, CHC Cost $90.64211

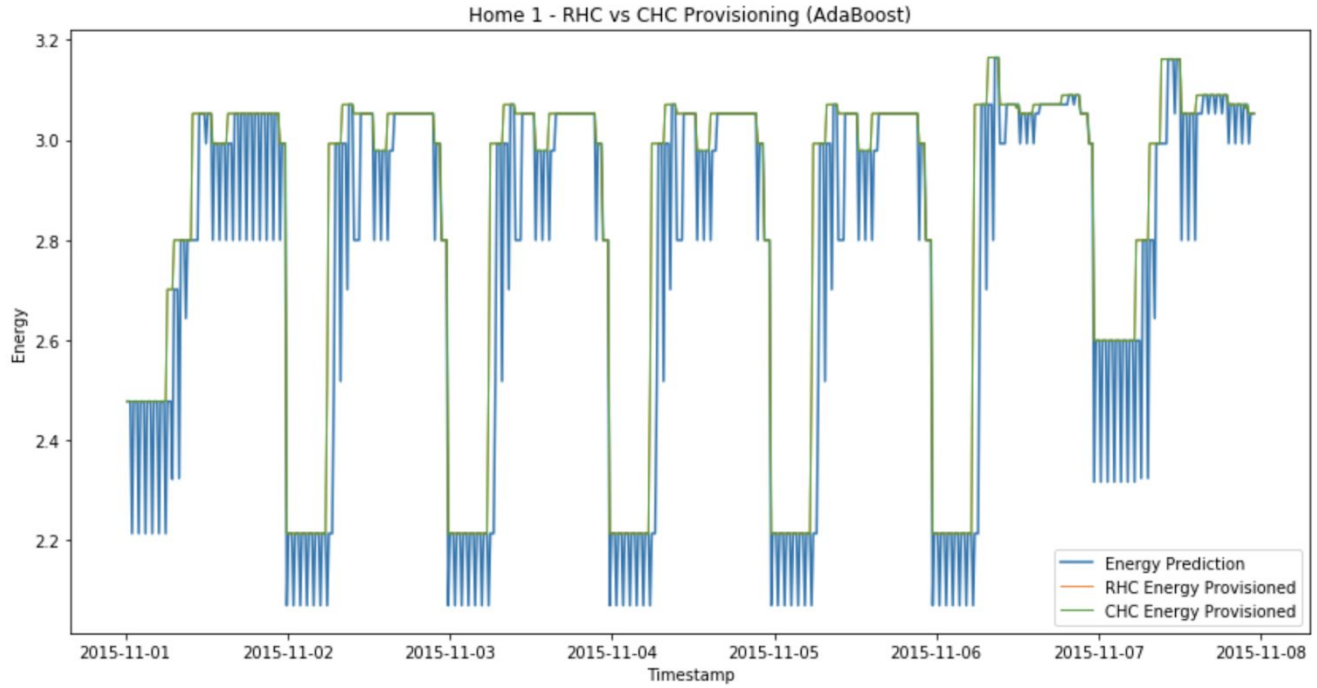

Figure 15: Home 1 - RHC (w = 5) vs CHC (w = 5, v = 2) using AdaBoost predictions
RHC Cost $204.60625, CHC Cost $204.605158

Below is the representation of Home 2 predictions for the RHC and CHC algorithms using the prediction values obtained from the ARIMA and AdaBoost models.

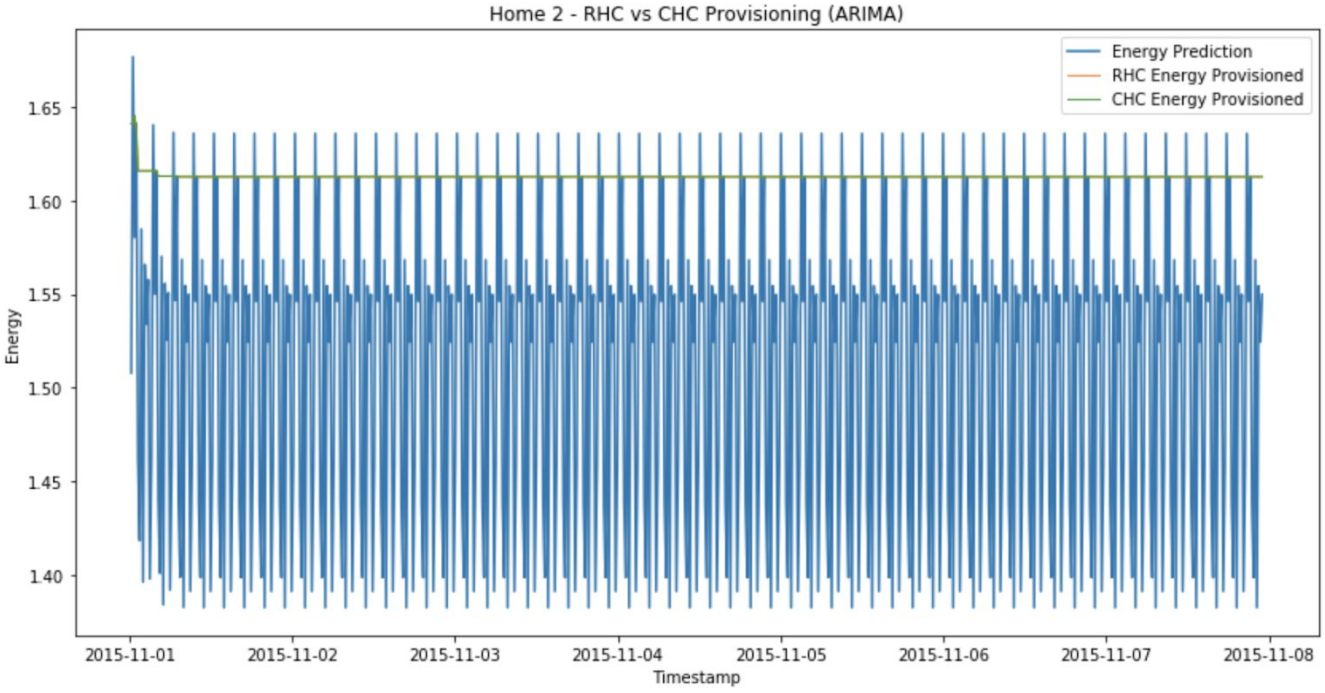

Figure 16: Home 2 - RHC (w = 12) vs CHC (w = 12, v = 2) using ARIMA predictions
RHC cost $122.99465, CHC Cost $122.99465

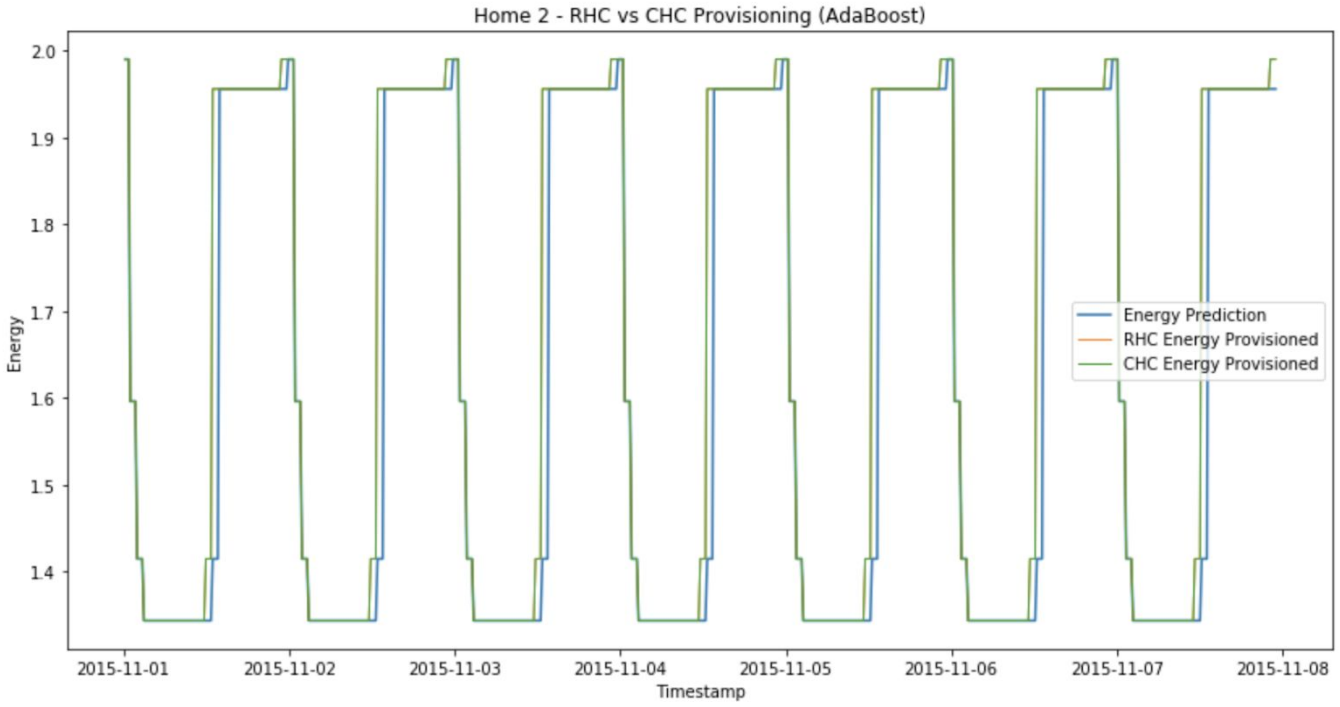

Figure 17: Home 2 - RHC (w = 5) vs CHC (w = 5, v = 2) using AdaBoost predictions
RHC Cost $111.21925, CHC Cost $111.21948

Below is the representation of Home 3 predictions for the RHC and CHC algorithms using the prediction values obtained from the ARIMA and AdaBoost models.

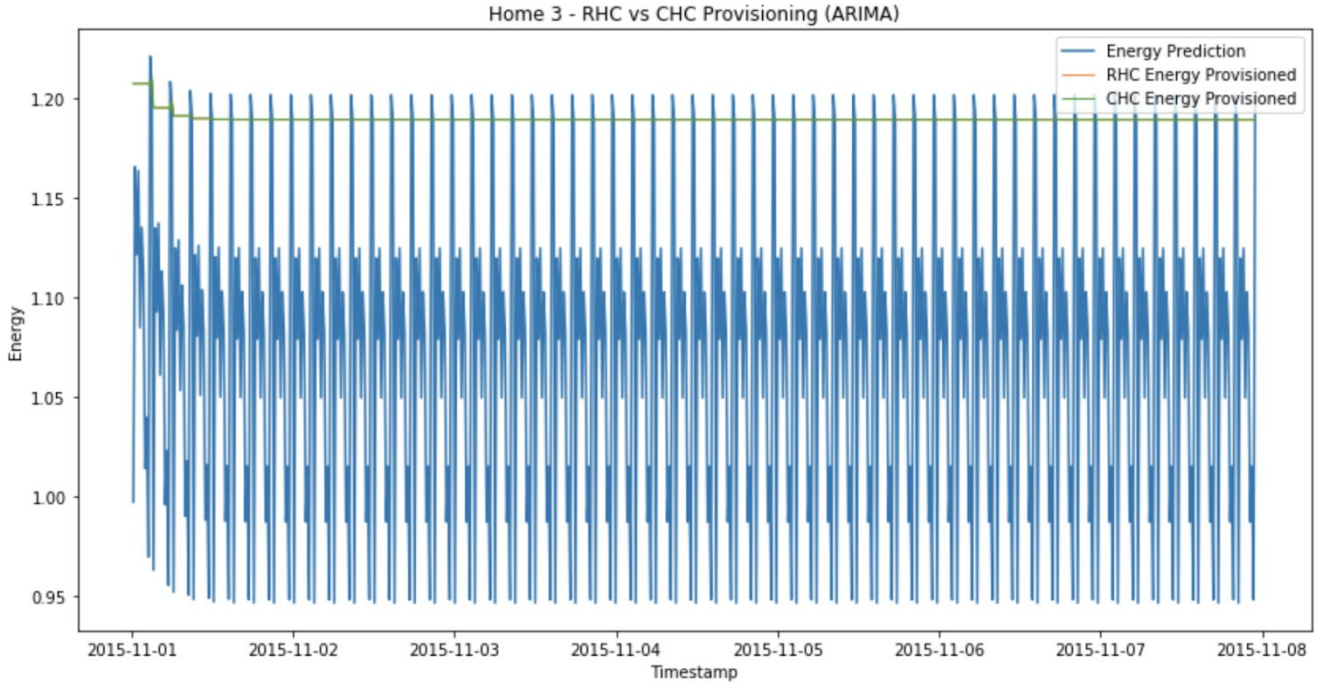

Figure 18: Home 3 - RHC (w = 12) vs CHC (w = 12, v = 2) using ARIMA predictions
RHC Cost $137.34040, CHC Cost $137.34040

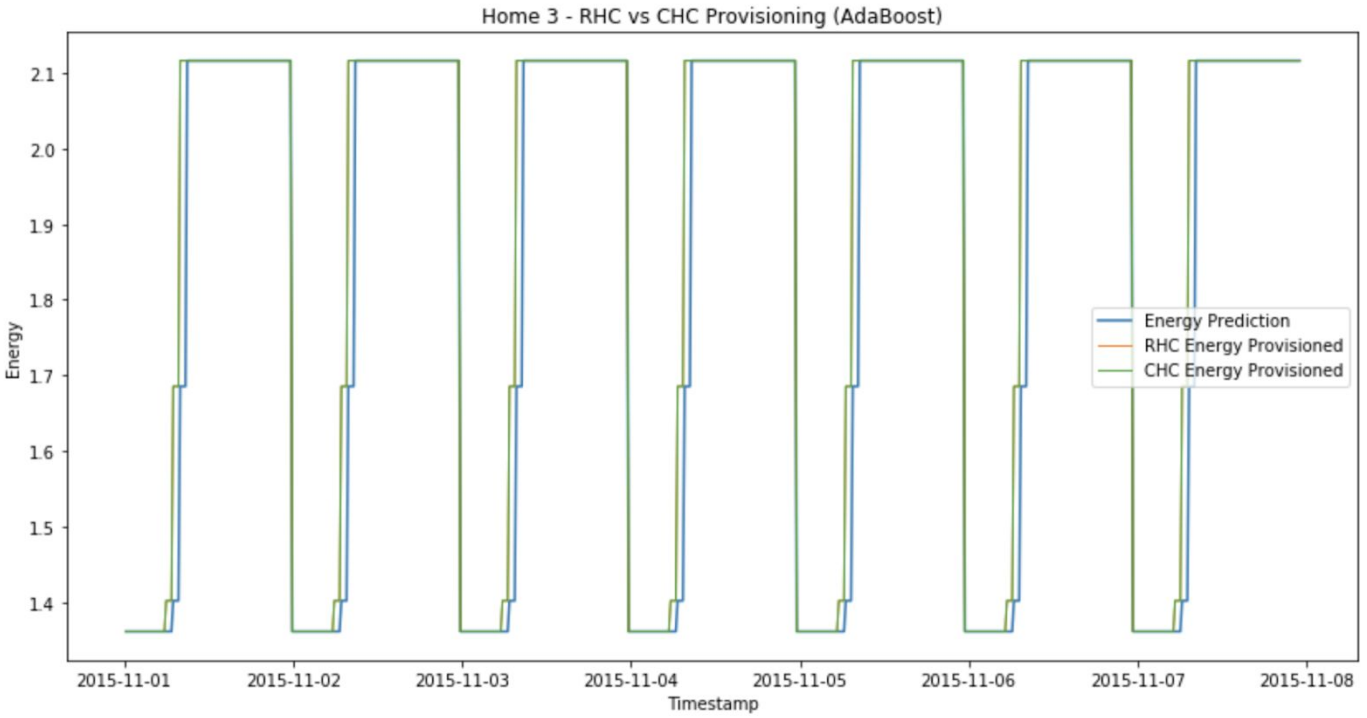

Figure 19: Home 3 - RHC (w = 5) vs CHC (w = 5, v = 2) using AdaBoost predictions
RHC Cost $81.787488, CHC Cost $81.787802

Observations from RHC and CHC:

1. The RHC and CHC produce almost similar provisioning values for most of the intervals on our timeline, though we achieve slightly better results results with CHC.
2. The RHC provisioning for arima predictions works best, i.e minimum cost, for window size = 12, whereas RHC for adaboost predictions works best for window size = 5.
3. CHC provisioning works best with commitment window size = 2 on top of best window size of 12 which we get from RHC provisioning.
4. The RHC and CHC provisioning for arima predictions generate almost straight line but for adaboost it follows the prediction patterns. This might be because arima generates accurate predictions within a very close range, whereas inputs from adaboost vary over a large range.
5. The cost generated from both RHC and CHC (~90) for the given week time period in case of ARIMA predictions is much lesser compared to OGD (~285). CHC (~89.9) is even slightly lower than RHC. This shows that CHC with Arima is the best combination of provisioning-prediction algorithm we have for the given cost function.

2) Sensitivity Analysis

This section explores the impact of changing parameters related to given control algorithms on the overall cost of the performance function.

a. **Impact of changing learning rate on OGD for houses 1, 2 and 3**
   Below are the plots of overall cost of function vs varying learning rate η from 0.1 to 1.0 for first 3 houses.

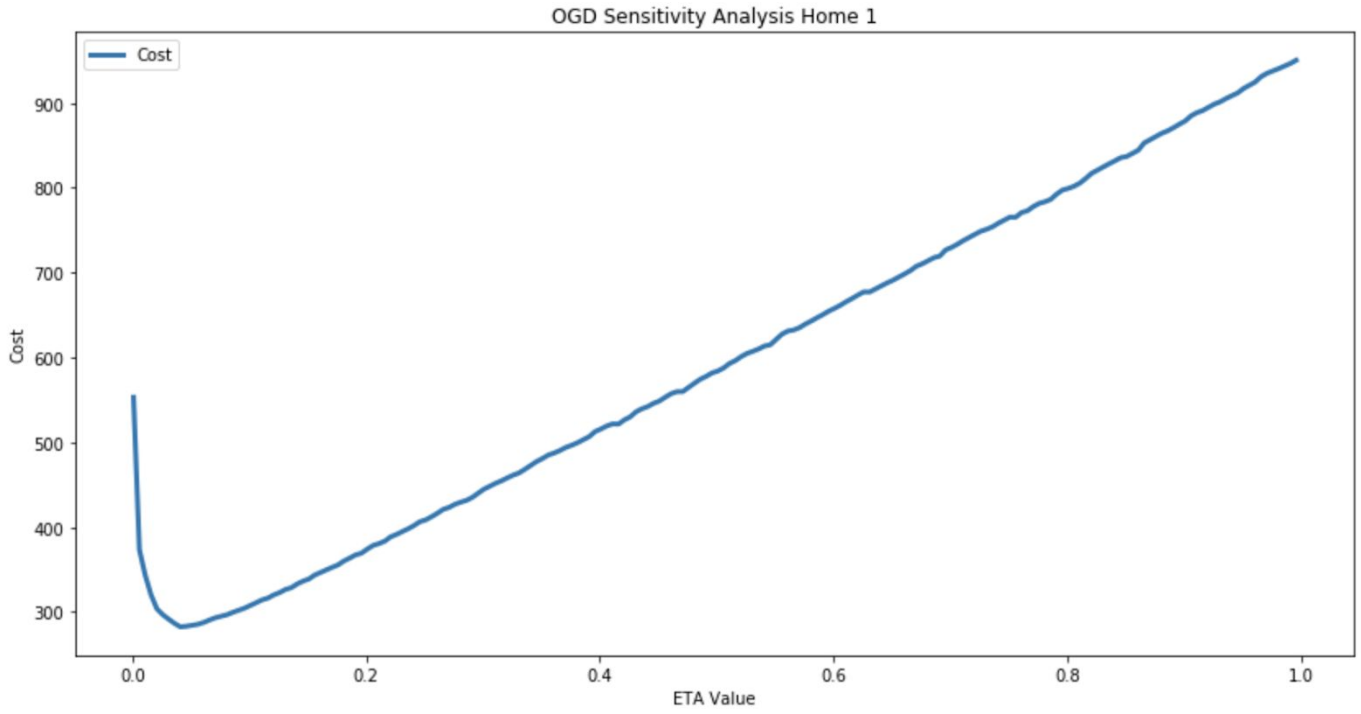

Figure 20: Home 1 - OGD Sensitivity Analysis η in range (0.1, 1.0)

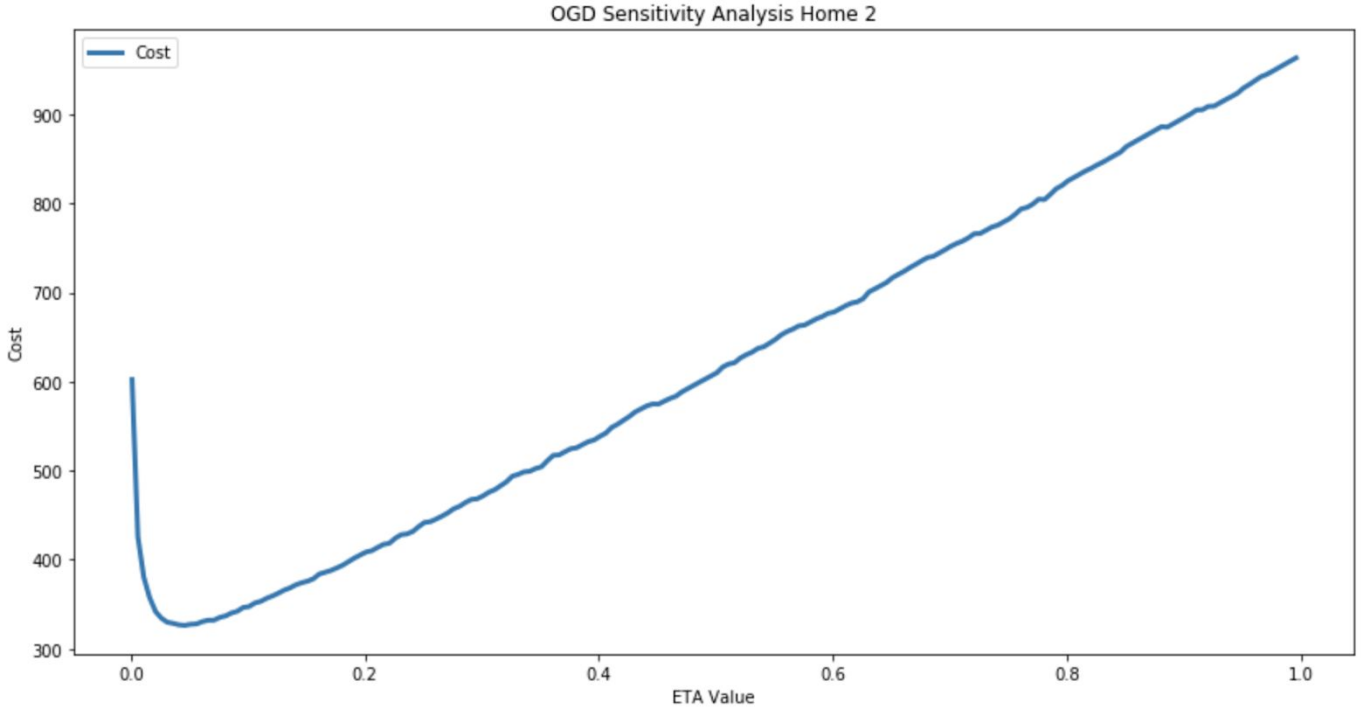

Figure 21: Home 2 - OGD Sensitivity Analysis η in range (0.1, 1.0)

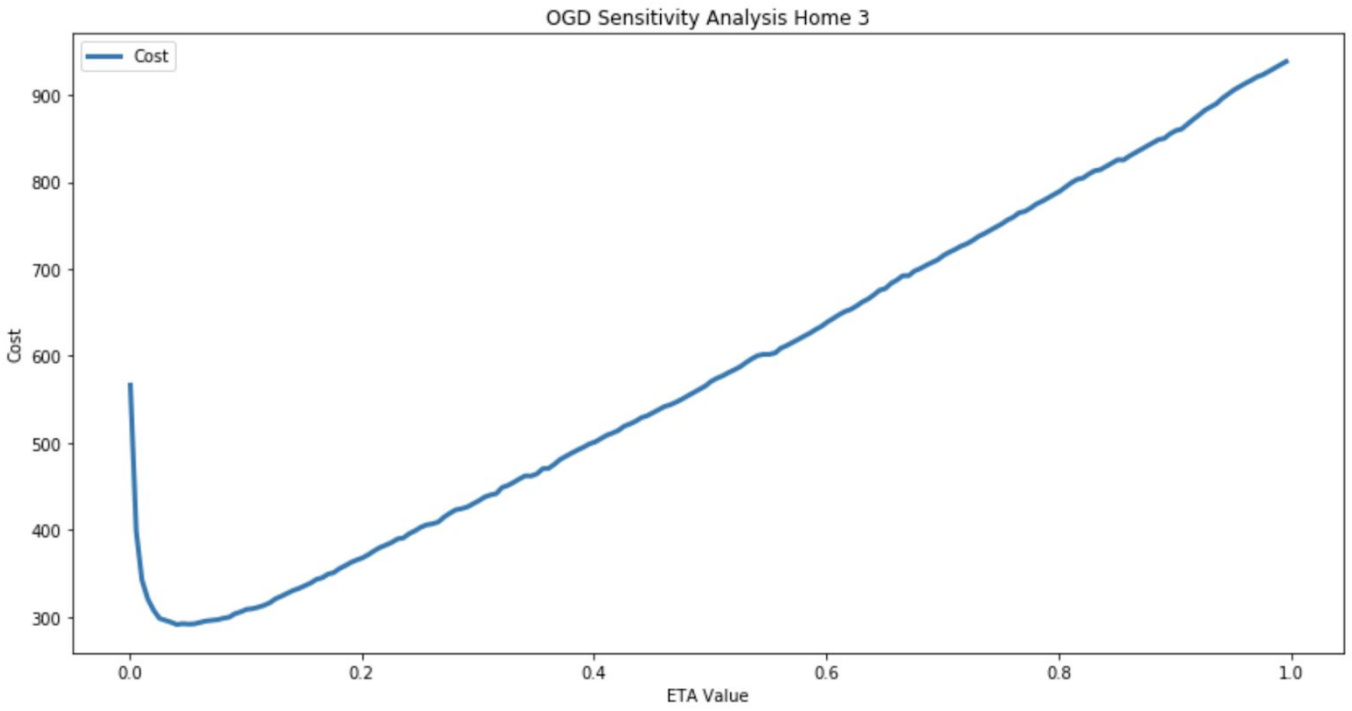

Figure 22: Home 3 - OGD Sensitivity Analysis η in range (0.1, 1.0)

Observation of varying learning rate on OGD:

1. From the OGD figures for all three houses shown above, it is evident that overall minimized cost of the OGD control algorithm occurs at learning rate η = 0.055.
2. The cost of the algorithm increases linearly as we increase the learning rate after 0.055 for all houses.
3. This also shows that a small value of learning rate is most effective in providing best provisioning results when compared to larger values. However, too much small values (~0) may also result in larger computed cost.

b. Impact of changing window size w for RHC

For both of arima and adaboost predictions for house 1, we try to vary window size in RHC control algorithm to find its impact on the overall provisioning cost. We vary the window size from 1 to 20.

Below are the plots of cost of RHC control algorithm against varying window size for house 1 arima and adaboost predictions.

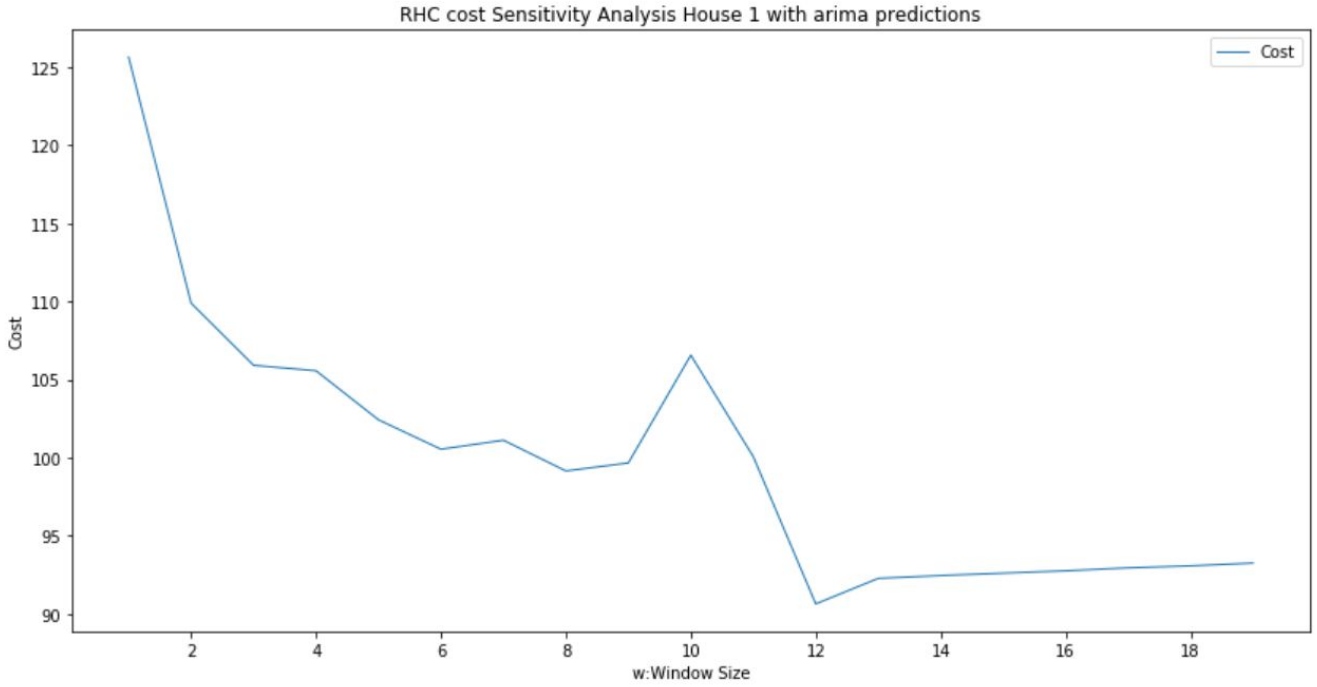

Figure 22: Home 1 - RHC cost Sensitivity Analysis for House 1 with ARIMA

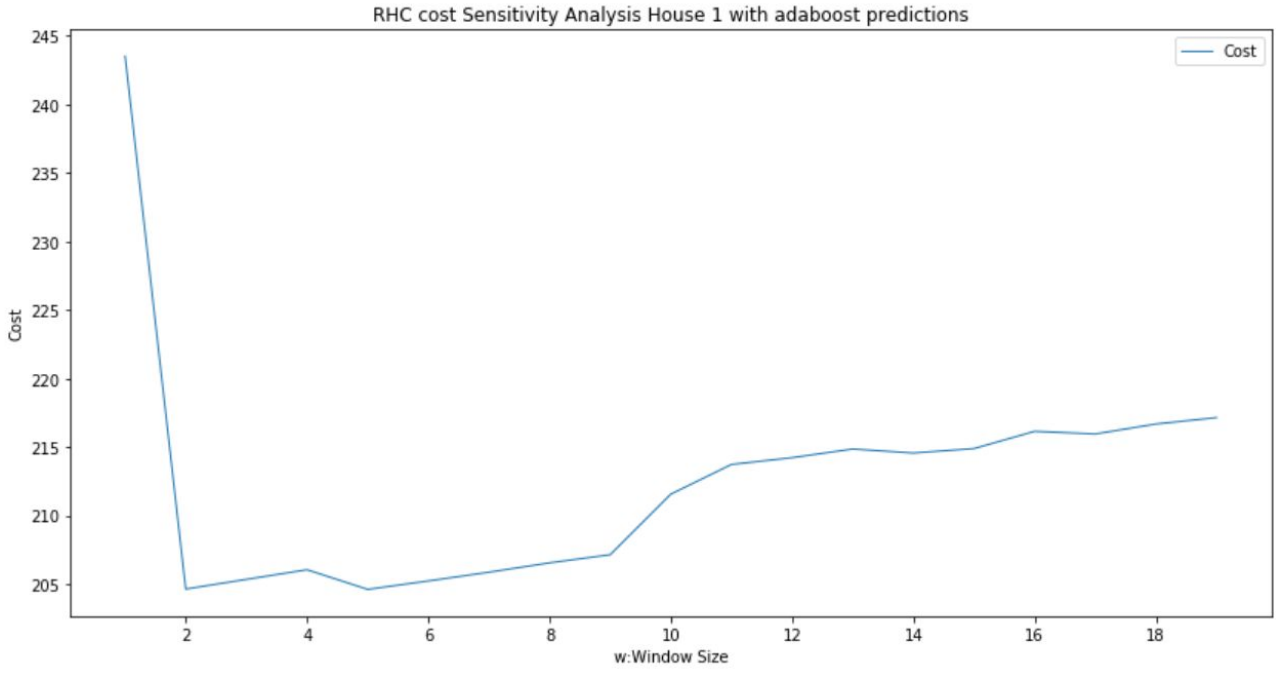

Figure 23: Home 1 - RHC cost Sensitivity Analysis for House 1 with AdaBoost

***Observation*:** After multiple trials, we found that, for arima prediction values, window size = 12 produces the minimum cost of the RHC (~90). We repeat our experiment similarly for adaboost predictions and find that the minimum cost of the algorithm occurs at window size = 5.

c. Impact of changing commitment window size v for CHC with w fixed

For both of arima and adaboost predictions for house 1, we try to vary commitment window size in CHC control algorithm from 1 to a fixed window size to find its impact on the overall provisioning cost. We vary the window size from 1 to 12 in case of arima predictions and from 1 to 5 in case of adaboost predictions, since these were the *w* which produced the best results in RHC for these prediction datasets.

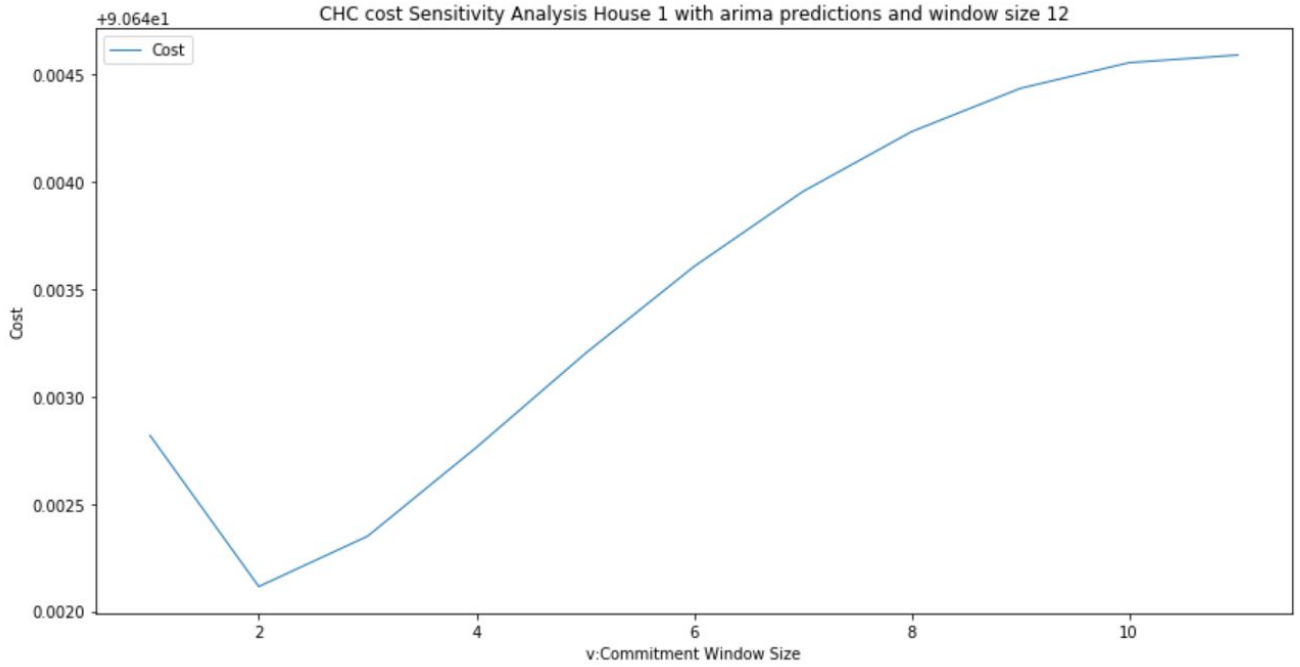

Figure 24: Home 1 -CHC cost Sensitivity Analysis for House 1 with ARIMA

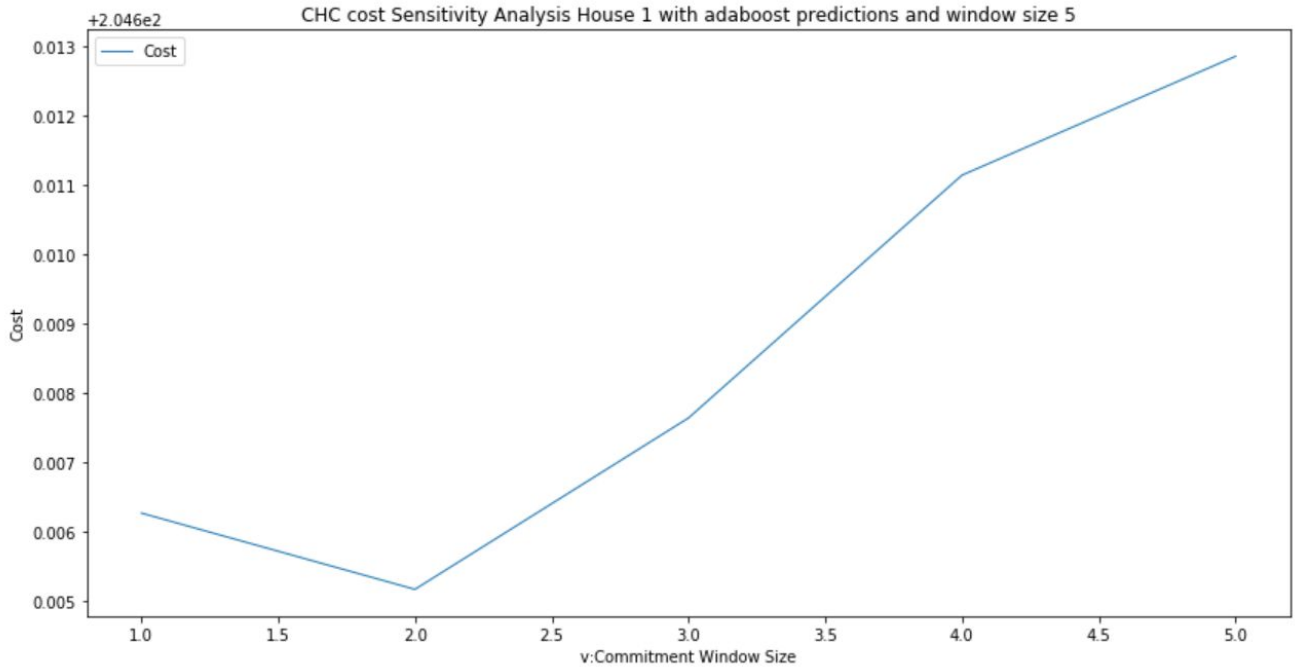

Figure 25: Home 1 - CHC cost Sensitivity Analysis for House 1 with AdaBoost

***Observation*:** After multiple trials with arima and adaboost predictions, we found that commitment window size v = 2 produces the minimum cost of the CHC (~89.9) for house 1. We use similar methodology to find the impact of varying w and v for other two homes and find similar results.

Task 3

Here we compare the energy provisioning done by each of the online algorithms with the offline solutions. Based on our observations from Task 1 and 2, we know that the energy provisioned by the Offline Dynamic Algorithm is the closest to the true demand and hence we can use Dynamic provisioning as our baseline to compare the the online algorithms. The data is for the week November 1, 2015 to November 7

- **Online Gradient Descent**

The graph below shows the energy provisioning done for Home 1 using the Offline algorithms against the Online Gradient Descent algorithm. The value of the learning rate i.e. η = 0.055.

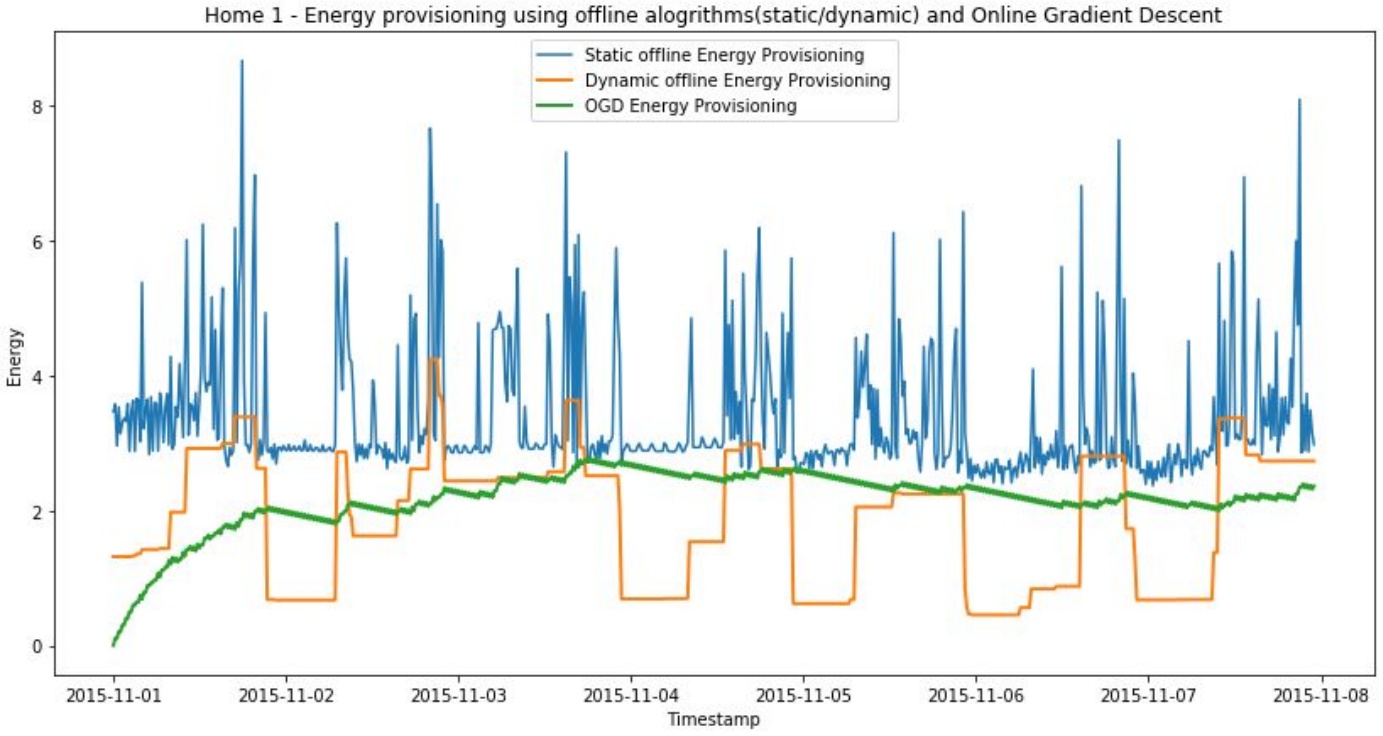

Figure 27: Home 1 -Energy Provisioning using offline algorithms(static/dynamic) and OGD

- **Receding Horizon Control (RHC)**

The graph below shows the energy provisioning done for Home 1 using the Offline algorithms against the RHC. The window size used with ARIMA prediction was 12 and the window size used with the AdaBoost prediction was 5.

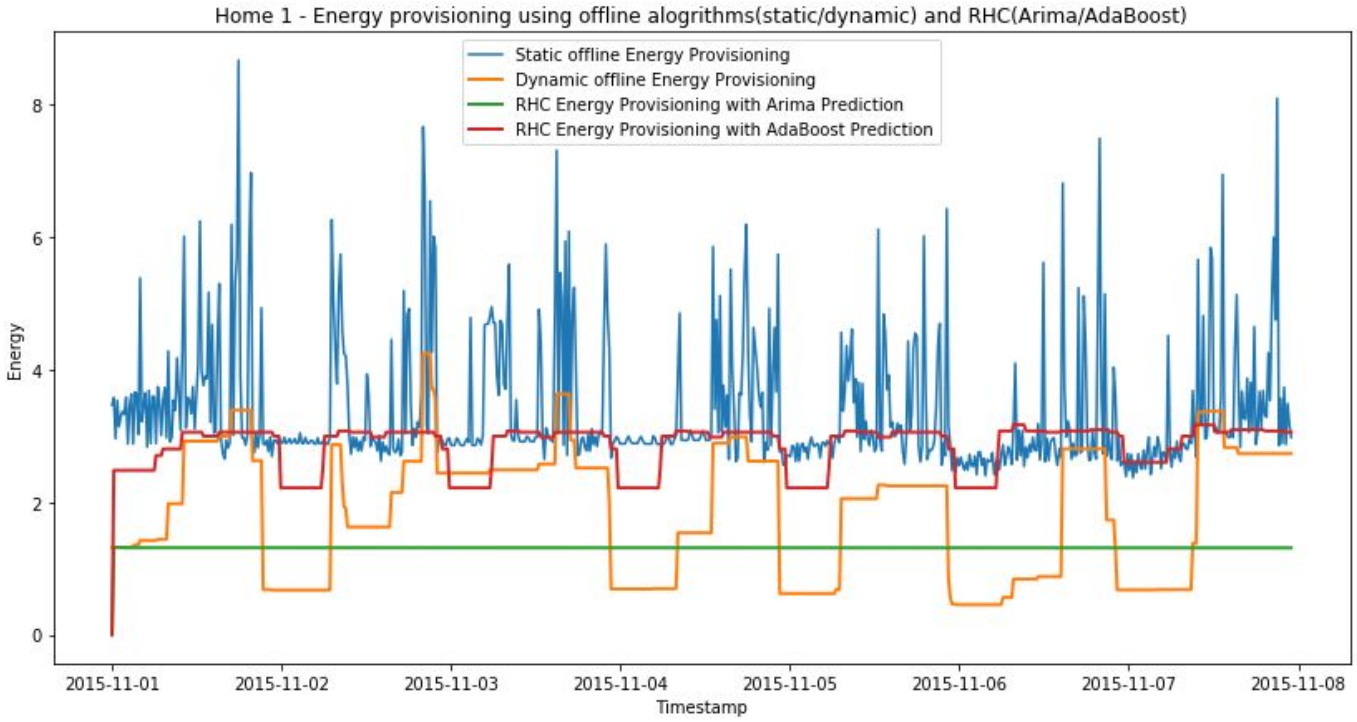

Figure 27: Home 1 -Energy Provisioning using offline algorithms(static/dynamic) and RHC(ARIMA/AdaBoost)

Observations:

1. With ARIMA prediction, there was very less variation amongst the consecutive values, and hence the graph for RHC with ARIMA prediction comes out to be a straight line.
2. RHC with AdaBoost prediction looks better and closer to the dynamic solution in terms of its behavior.

- **Commitment Horizon Control (CHC)**

The graph below shows the energy provisioning done for Home 1 using the Offline algorithms against the CHC with both ARIMA and AdaBoost predictions.
For both the cases, the window size(w) used was 10 with a commitment window size (v) of 5

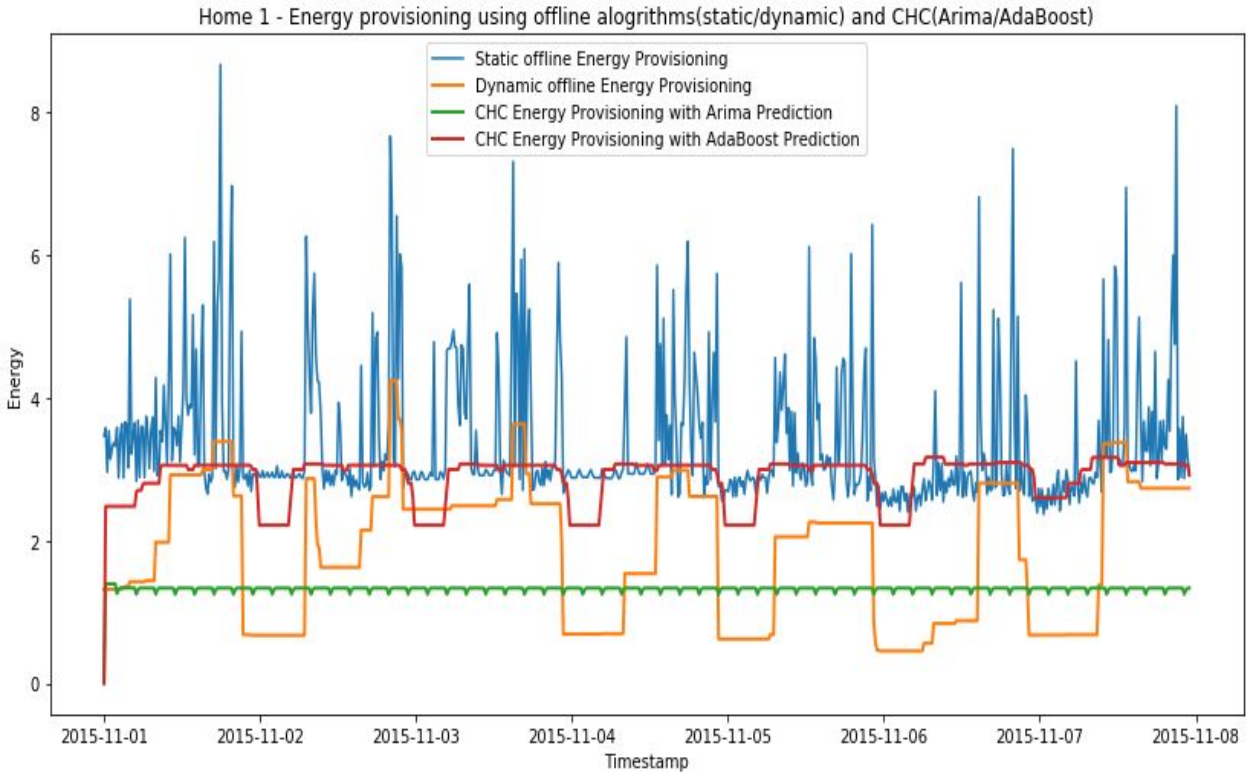

Figure 28: Home 1 -Energy Provisioning using offline algorithms(static/dynamic) and CHC(ARIMA/AdaBoost)

Observations:

1. For CHC with ARIMA, though we do not see a straight line as opposed to RHC, we see some consistency in the behavior.
2. For CHC with Adaboost, the behavior looks similar to RHC with Adaboost.
3. For CHC as well, the results look better with the AdaBoost prediction.

Cost comparison of the Online Algorithms with the Offline static and dynamic solutions.

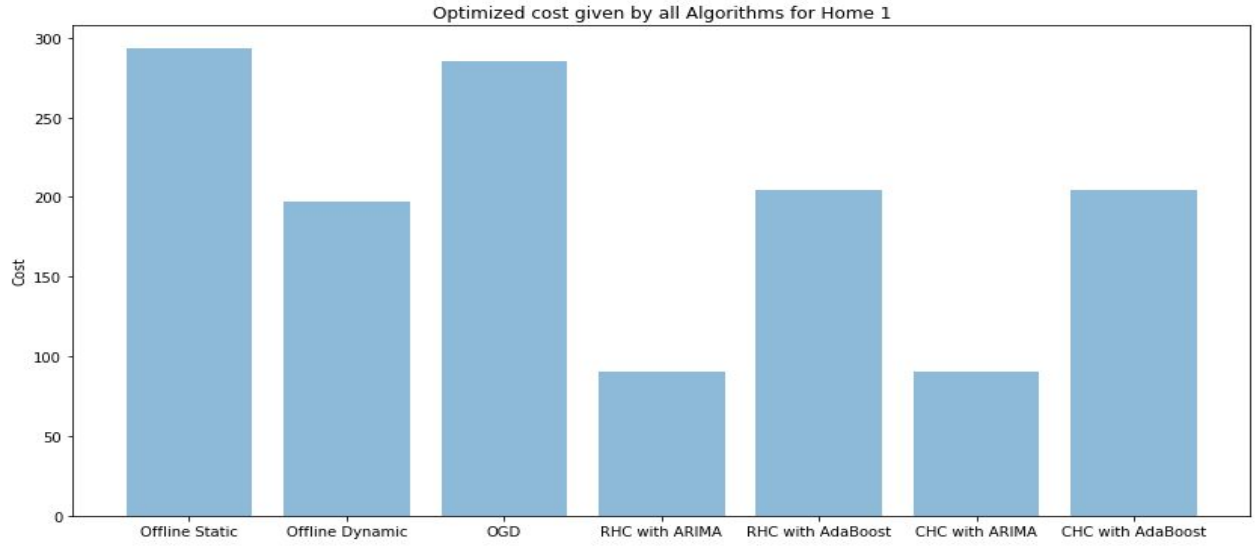

Figure 29: Home 1 - Optimized cost comparison for all the algorithms

Observations:

1. From the above bar graph, we can see that the cost given by CHC with Arima($90.64211) is least of all the algorithms.
2. Though RHC with ARIMA ($90.64281) looks similar to CHC with ARIMA, there is a minute difference in the cost values of both.
3. The Objective function cost for the CHC with ARIMA prediction algorithm was minimum and hence this is our best combination.
4. The cost given by the offline algorithms is higher than the online algorithms.

Task 4

From our observations in Task 3, we know that we get the minimum cost using **CHC (w = 12, v = 2) with ARIMA prediction algorithm**. Hence, that is our best combination.

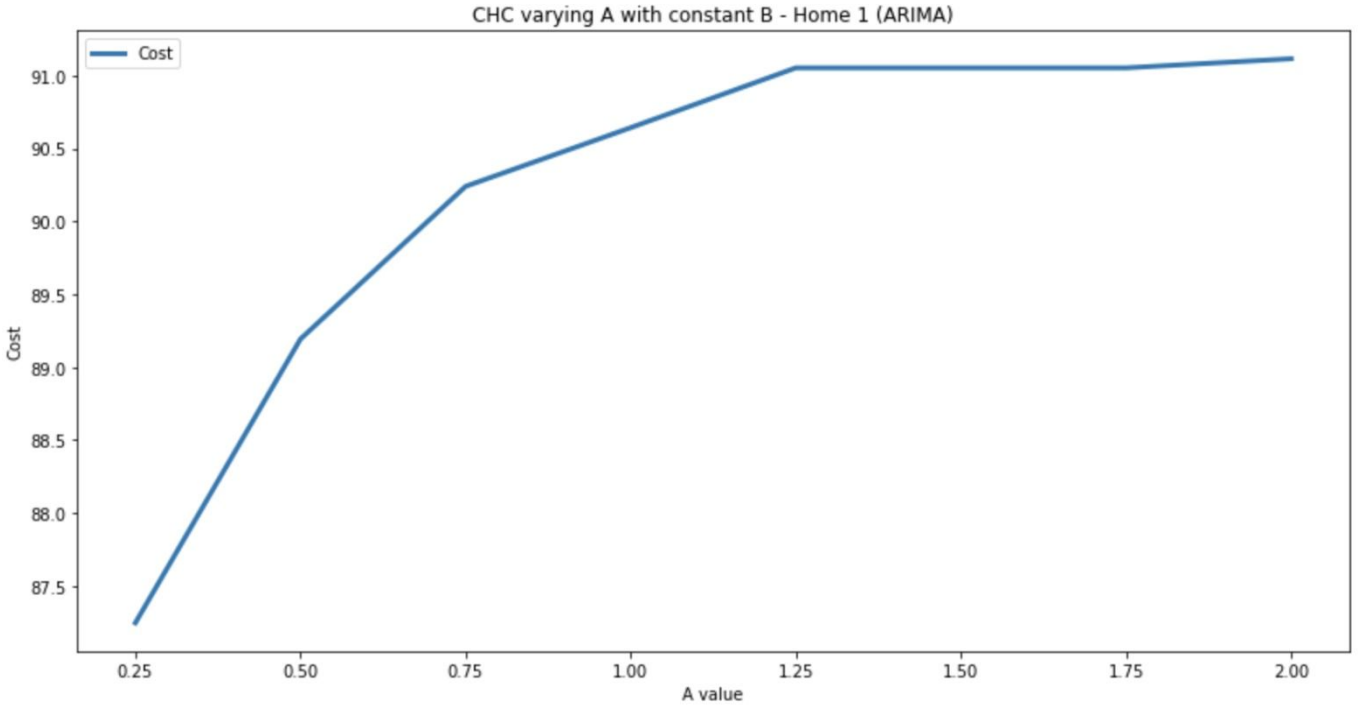

Figure 30: Home 1 - CHC (w = 12, v = 2) with varying A and constant B with ARIMA predictions

Observations:

1. We varied “a” from $0.25/kw per 15 minute to $2/kw per 15 minute while keeping “b” constant at $1/kw per 15 minute and we saw a gradual increase in cost function with increasing “a” value with a saturation point near “a” = $1.25/kw per 15 minute

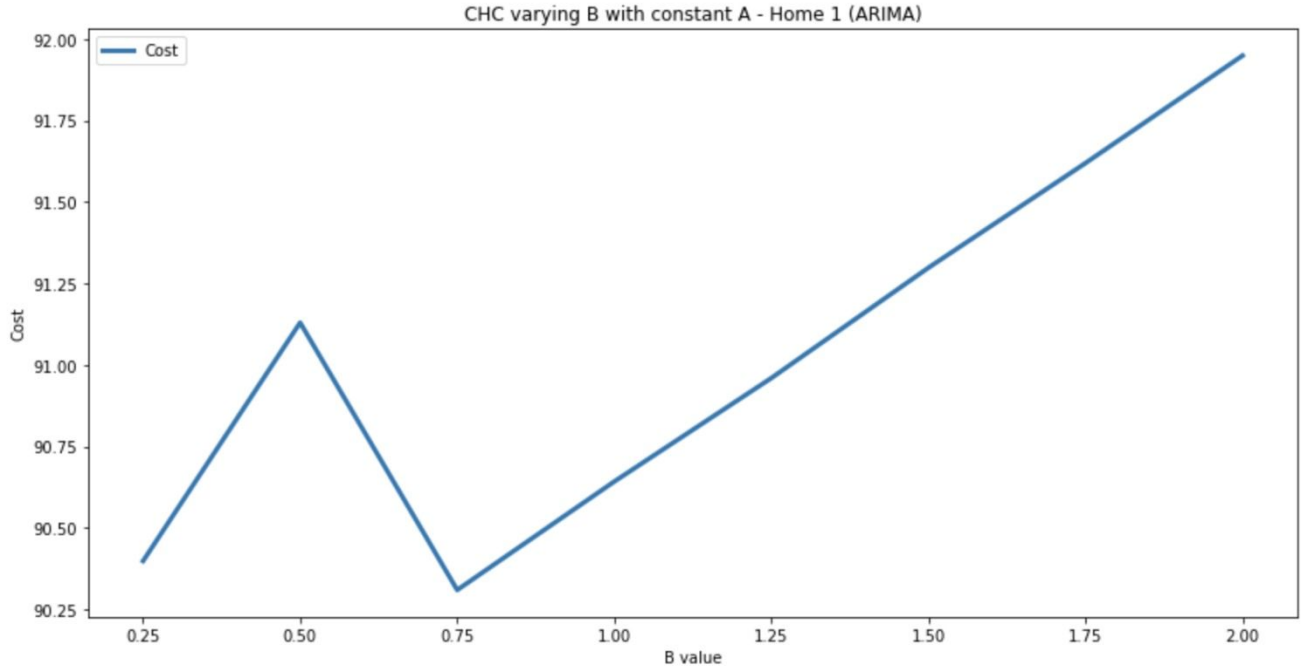

Figure 31: Home 1 - CHC (w = 12, v = 2) with varying B and constant A with ARIMA predictions

Observations:

1. We varied “b” from $0.25/kw per 15 minute to $2/kw per 15 minute while keeping “a” constant at $1/kw per 15 minute and we saw an increase in cost function with increasing “b” value until $0.5/kw per 15 minute.
2. Between $0.5/kw per 15 minute and $0.5/kw per 15 minute, we observed a constant dip in the cost value and further from there we saw a linear increase in cost until “b” reached $2/kw per 15 minute.
3. We did not observe a saturation point for the cost function with varying “b”.

Task 5

In this task, we are computing the x values for the week November 1, 2015 to November 7, 2015 by using the Deterministic and the Randomized approach.All the computations done are for Home 1.

- **Deterministic Approach**

1. In this approach, we pick all the five online algorithms, OGD, RHC with ARIMA, RHC with AdaBoost, CHC with ARIMA, CHC with AdaBoost and switch an algorithm on every hour i.e. K = 4 in Round Robin manner.
2. Starting with t = 1, we run OGD till t= 4, then switch to RHC with ARIMA at t = 5 and run it till t = 8 and so on until CHC with Adaboost starting at t = 16 and running till t = 20 and then repeat this cycle for the further timeslots till t = 672.
3. We compute the final cost over all the X values using the objective function. The cost obtained was $664.58389
4. Below is the code snippet and the plot of energy demand vs energy provisioned for the deterministic approach.

```
x_best_d = [0]
cost_d = 0
for t in range(1,35):
    x_best_d , cost_d = OGD(t , x_best_d, cost_d)
    x_best_d , cost_d = RHC_ARIMA(t + 4 , x_best_d, cost_d)
    x_best_d , cost_d = CHC_ARIMA(t + 8 , x_best_d, cost_d)
    x_best_d , cost_d = RHC_ADABOOST(t + 12 , x_best_d, cost_d)
    x_best_d , cost_d = CHC_ADABOOST(t + 16 , x_best_d, cost_d)
```

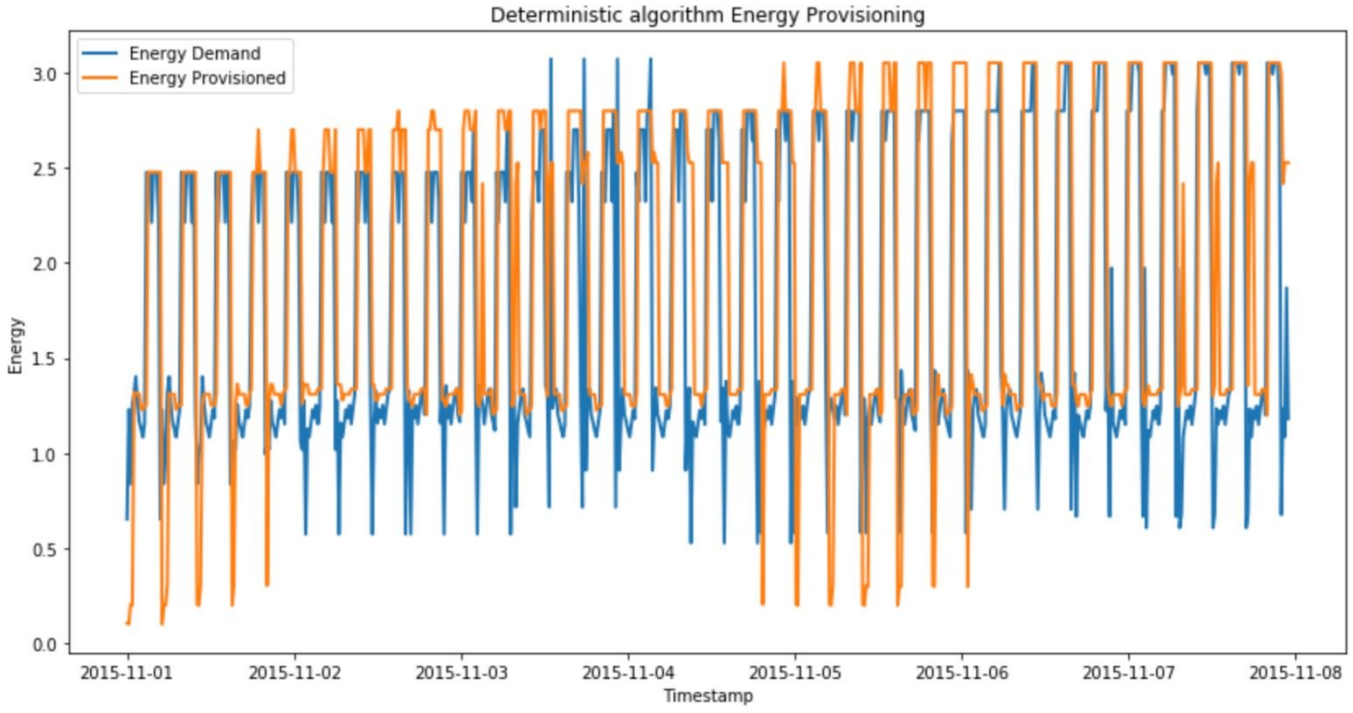

Figure 32: Home 1 - Deterministic Algorithm Energy provisioning

- **Randomized Approach**

1. In this approach, we use the same algorithms as above, but not in the Round Robin manner.
2. Here the algorithms are called based on the weights assigned to each of them. These weights are assigned based on the costs we obtained from them in the previous tasks. The algorithm with the least cost will have the highest weight. The sum of the weights of all the algorithms is 1.
3. Below mentioned are the algorithms with their corresponding weights
    - CHC with ARIMA : 0.3
    - RHC with ARIMA : 0.25
    - CHC with AdaBoost : 0.2
    - RHC with AdaBoost : 0.15
    - OGD : 0.1

4. The cost obtained using the Random approach is $458.293186
5. Below is the code snippet and the plot of energy demand vs energy provisioned for the Randomized approach.

```
import random
my_list = ['OGD'] * 10 + ['RHC_ARIMA'] * 25 + ['CHC_ARIMA'] * 30 + ['RHC_ADABOOST'] * 15 +  ['CHC_ADABOOST'] * 20
x_best_r = [0]
cost_r = 0
for t in range(1,35):
    x_best_r , cost_r = globals()[random.choice(my_list)](t , x_best_r, cost_r)
    x_best_r , cost_r = globals()[random.choice(my_list)](t + 4 , x_best_r, cost_r)
    x_best_r , cost_r = globals()[random.choice(my_list)](t + 8 , x_best_r, cost_r)
    x_best_r , cost_r = globals()[random.choice(my_list)](t + 12 , x_best_r, cost_r)
    x_best_r , cost_r = globals()[random.choice(my_list)](t + 16 , x_best_r, cost_r)
```

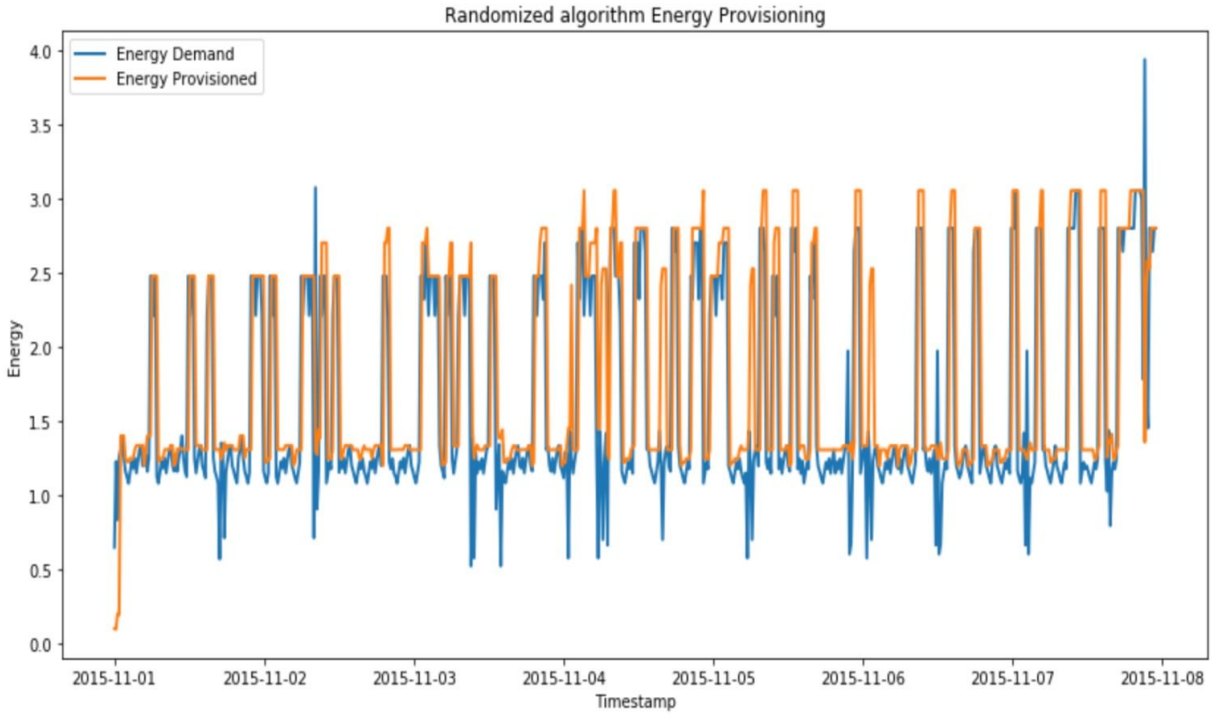

Figure 33: Home 1 - Randomized Algorithm Energy provisioning