# CSE 551 HW2 Report

By Group 11
Akanksha Kale (SBU ID: 113788594)
Akhila Juturu (SBU ID: 114777498)
Mandar Laxmikant Mahajan (SBU ID: 113276053)

## Overview

We use the same dataset with load demand for 3 users from the previous assignment. For certain algorithms, we also have to use predicted data from the previous assignment. We had trained two models in a previous assignment: ARIMA (statistical model) and LSTM (deep learning model). Accordingly, we have two sets of predictions for each of the 3 users.

We try out the following algorithms for electricity provisioning:

- Offline Static Optimization
- Offline Dynamic Optimization
- Online Gradient Descent
- Receding Horizon Control
- Commitment Horizon Control

The performance of these algorithms is measured by the objective function given by:

$$\sum_{t=1}^{T} p(t)x(t) + a * \max\{0, y(t) - x(t)\} + b|x(t) - x(t-1)|$$

Objective function consists of three terms:

- First term: $p(t)x(t)$
  This term represents the price at which we are buying the electricity. **p(t)** is the cost of electricity. We have used a constant value for p(t) = $ 0.40 / kWh.
  As our time interval is per 30 minutes, and not hourly, we pass p = 0.20
- Second term: $a * \max\{0, y(t) - x(t)\}$
  This term represents the penalty that we would incur if we have not purchased enough electricity with respect to the true value. This would mean our electricity provisioning is not able to meet the demands. We have used a = $4 / kWh.
  As our time interval is per 30 minutes, and not hourly, we pass a = 2

- Third term: $b|x(t) - x(t-1)|$
  This term represents the penalty if our electricity provisioning algorithm is not flat enough. We have used b = $4 / kWh. As our time interval is per 30 minutes, and not hourly, we pass b = 2

According to the instructions, we have not used predictions for offline optimization problems and the online gradient descent algorithm.

We have **T = 672**, which represents two week data from Dec 1 - Dec 14 in 30 minute intervals.

## Data massaging

In the previous assignment, we have predicted hourly intervals. However, we require predictions in 30 minute intervals for this assignment. Thus, we have resampled the predictions as follows:

Before:

| datetime | House overall [kW] | forecast |
|---|---|---|
| 2014-11-07 10:00:00 | 0.923507 | 0.779759 |
| 2014-11-07 11:00:00 | 0.578251 | 0.927103 |
| 2014-11-07 12:00:00 | 0.473563 | 0.755461 |
| 2014-11-07 13:00:00 | 0.511153 | 0.668223 |
| 2014-11-07 14:00:00 | 0.956069 | 0.688386 |
| ... | ... | ... |
| 2014-12-31 19:00:00 | 2.827218 | 3.380369 |
| 2014-12-31 20:00:00 | 2.742687 | 2.836219 |
| 2014-12-31 21:00:00 | 2.340254 | 2.729116 |
| 2014-12-31 22:00:00 | 1.755305 | 2.445897 |
| 2014-12-31 23:00:00 | 1.995542 | 1.947730 |

1310 rows × 2 columns

After:

| | datetime | House overall [kW] | forecast |
|---|---|---|---|
| **1132** | 2014-12-01 00:00:00 | 0.533483 | 0.481164 |
| **1133** | 2014-12-01 00:30:00 | 0.533483 | 0.481164 |
| **1134** | 2014-12-01 01:00:00 | 0.636498 | 0.500625 |
| **1135** | 2014-12-01 01:30:00 | 0.636498 | 0.500625 |
| **1136** | 2014-12-01 02:00:00 | 0.448014 | 0.557183 |
| **...** | ... | ... | ... |
| **1799** | 2014-12-14 21:30:00 | 1.117833 | 1.213744 |
| **1800** | 2014-12-14 22:00:00 | 1.255763 | 1.227928 |
| **1801** | 2014-12-14 22:30:00 | 1.255763 | 1.227928 |
| **1802** | 2014-12-14 23:00:00 | 0.538909 | 1.257196 |
| **1803** | 2014-12-14 23:30:00 | 0.538909 | 1.257196 |

672 rows × 3 columns

## Offline Static Optimization

Offline static optimization is a convex optimization problem where we predict a constant provisioning value for the entire time period.

For the purposes of convex optimization, we have utilized the CVXPY library.

In offline static optimization, as the provisioning value is constant for the entire time period, the third term in the cost function vanishes (as x(t) - x(t-1) is always zero). Thus, we are then left with the following:

$$\sum_{t=1}^{T} p(t)x(t) + a * \max\{0, y(t) - x(t)\}$$

As x(t) is constant for this optimization, we only have a single variable that we need to predict.

We encode the above equation into a CVXPY problem as follows:

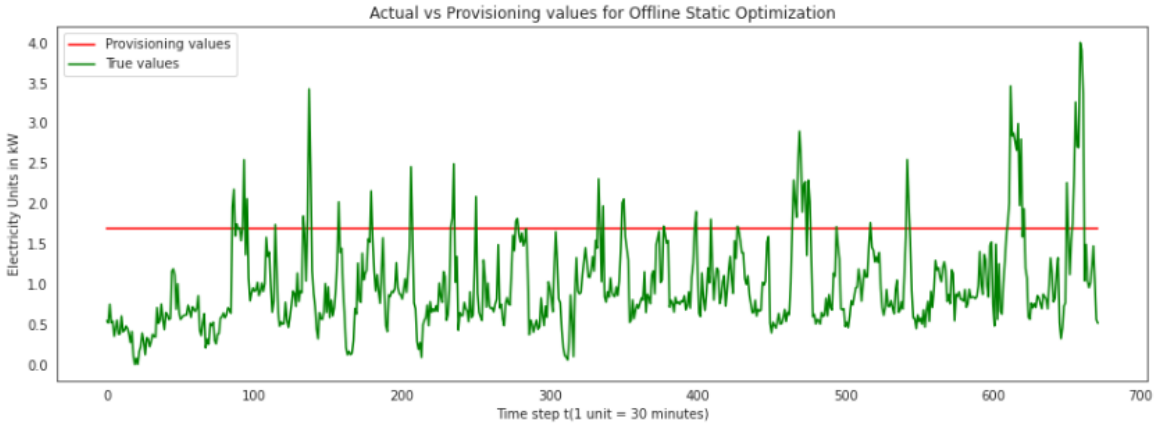
```python
def offline_static_opt(p, a, y):
  x = cp.Variable()
  f = 0

  for t in range(len(y)):
    f += p * x + a * cp.maximum(0, y[t] - x)

  prob = cp.Problem(cp.Minimize(f))
  prob.solve()

  return prob.value, x.value
```

As we see above, the parameter **b** is not used for offline static optimization as the third term from the cost function is always 0.

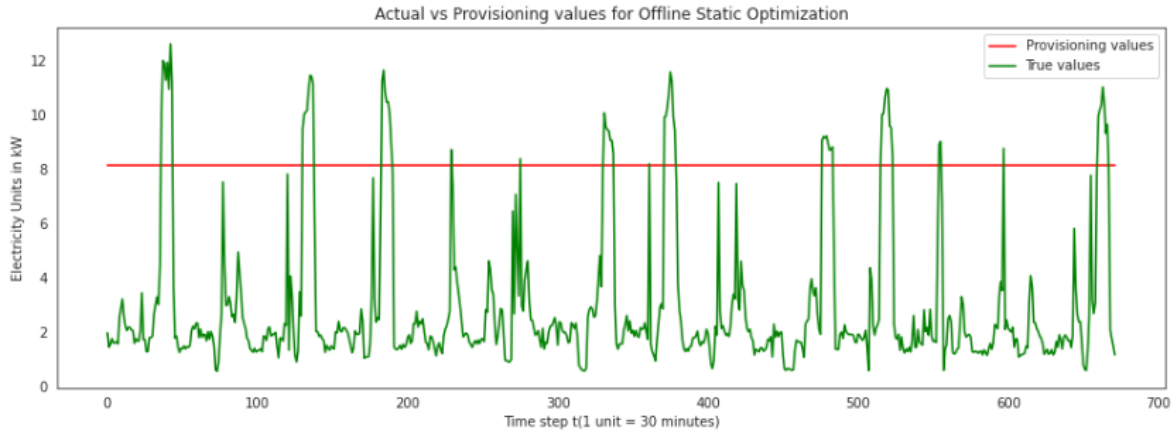
Running offline static optimization using the above algorithm for home 1, 2 and 3, we get the following results:

| | Offline static optimization cost($) | Provisioning value(kW) |
|---|---|---|
| Home 1 | 303.55 | 1.68 |
| Home 2 | 1348.84 | 8.146 |
| Home 3 | 4775.3275 | 24.902 |

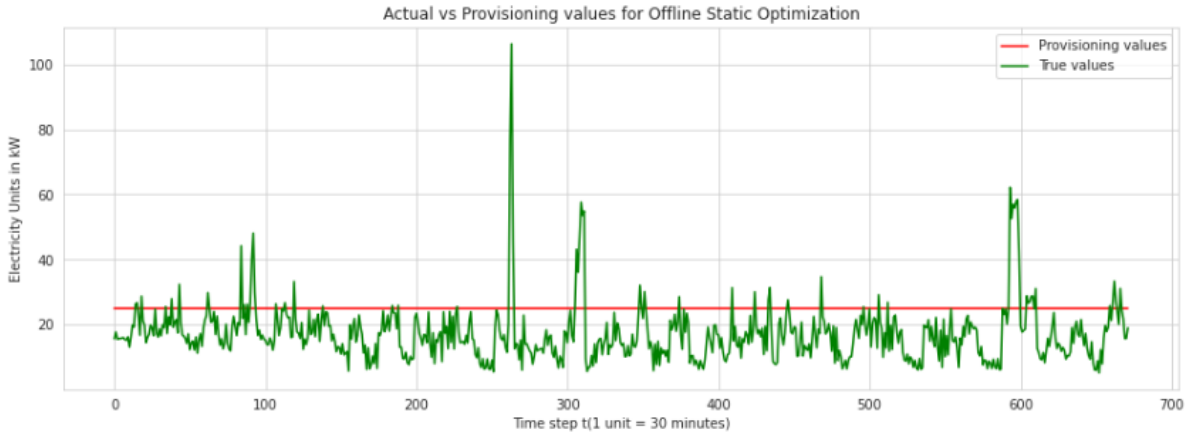
Home 1:




Home 2:




Home 3:




## Offline Dynamic Optimization

Offline dynamic optimization is a convex optimization problem where we predict a variable provisioning value for the given time period.

The objective function that we optimize is

$$\sum_{t=1}^{T} p(t)x(t) + a * \max\{0, y(t) - x(t)\} + b|x(t) - x(t-1)|$$

Here, we have to take x(t) as an array of 672 length, as we will have a provisioning value for each instance. We encode the above optimization into a CVXPY problem as follows:

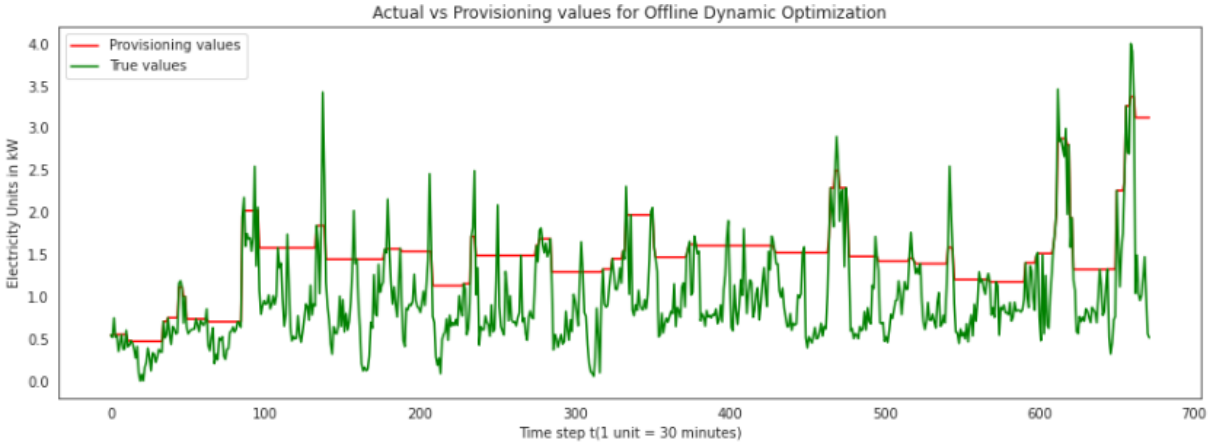
```
def offline_dyn_opt(p, a, b, y):
  x = cp.Variable(len(y))

  # cost for first time interval
  f = (p * x[0]) + (a * cp.maximum(0, y[0] - x[0])) + b * cp.abs(x[0] - 0)

  for t in range(1, len(y)):
    f += (p * x[t]) + (a * cp.maximum(0, y[t] - x[t])) + (b * cp.abs(x[t] - x[t - 1]))

  prob = cp.Problem(cp.Minimize(f))
  prob.solve()

  return prob.value, x.value
```

We have assumed x(0) = 0. Thus, we have calculated the cost corresponding to the first time interval outside the for-loop.

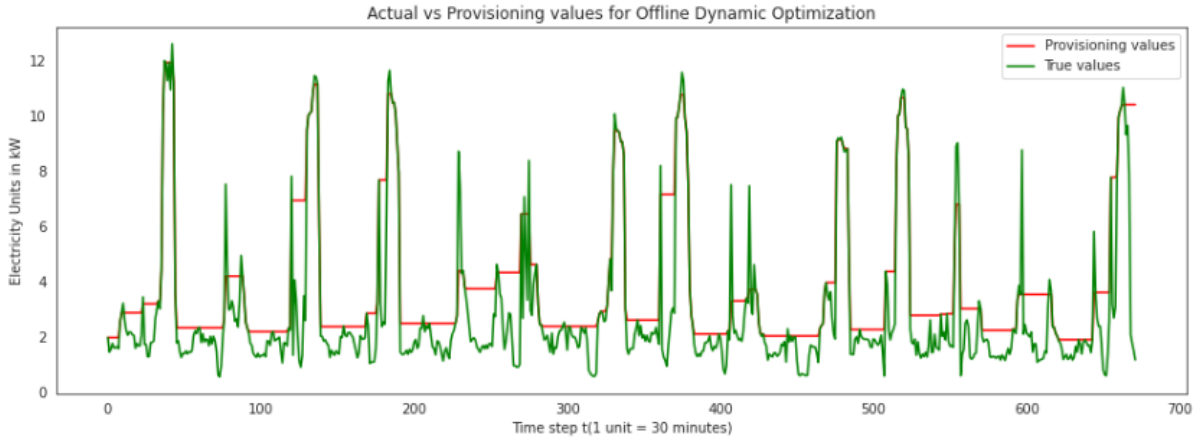
Running offline dynamic optimization using the above algorithm for home 1, 2 and 3, we get the following results:

| | Offline dynamic optimization cost($) |
|---|---|
| Home 1 | 254.36 |
| Home 2 | 906.56 |
| Home 3 | 4233.488 |

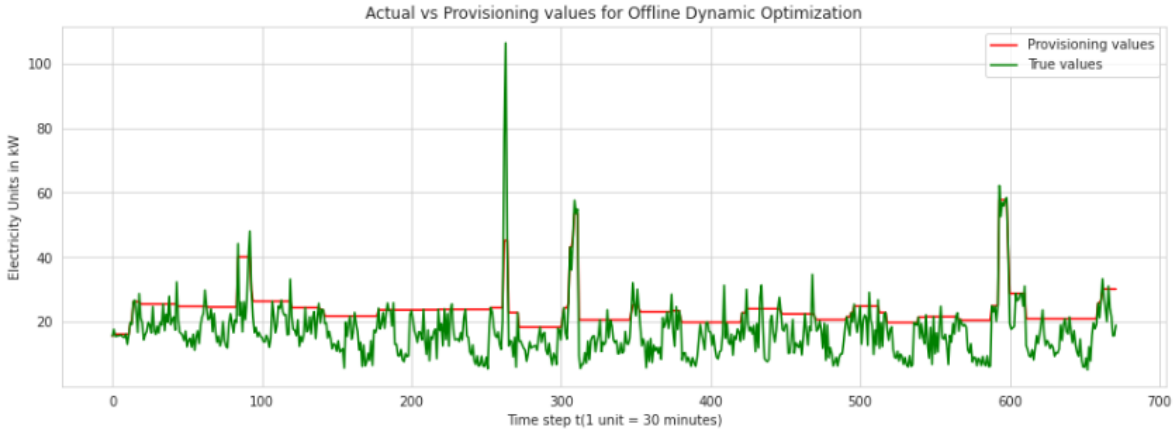
Home 1:



Home 2:




Home 3:




## Comparing Offline Dynamic and Static Optimization

For all the three homes, we can see that offline dynamic optimization is able to get a lower cost with the objective function. This is clearly because the dynamic optimization solution is able to minimize cost incurred due to the second term in the objective function (difference between true value and provisioning value).

| | Offline static optimization cost | Offline dynamic optimization cost |
|---|---|---|
| Home 1 | 303.55 | 254.36 |
| Home 2 | 1348.84 | 906.56 |

| | | |
|---|---|---|
| Home 3 | 4775.3275 | 4233.488 |

## Online Gradient Descent

Online gradient descent algorithm is similar to its famous counterpart: the stochastic gradient descent algorithm.

The equation for online gradient descent is as follows:

$$x(t) = x(t-1) - \alpha \cdot \frac{\partial J(x(t))}{\partial x(t)}$$

Here, $\alpha$ is the learning rate, and $\frac{\partial J(x(t))}{\partial x(t)}$ is the partial derivative of the cost function with respect to x(t), also called the gradient of the function.

We will initially fix our learning rate to some value, say 0.02. Later, we will iterate over a range of learning rates to find the best learning rate.

### Gradient calculation:

We need to take partial derivative of our cost function with respect to x(t):

$$\frac{\partial(p(t)x(t)+a\cdot max\{0,y(t)-x(t)\}+b\cdot|x(t)-x(t-1)|)}{\partial(x(t))}$$

To understand how we will calculate the gradient, let's look at each of the three terms in the above equation separately:

1. $$\frac{\partial(p \cdot x(t))}{\partial(x(t))}$$

   Evaluating the above is trivial, as we have chosen **p** to be a constant. Thus, the above term is equal to **p**.

2. $$\frac{\partial(a \cdot max\{0, y(t) - x(t)\})}{\partial(x(t))}$$

   Here, if y(t) < x(t), the above gradient evaluates to 0. However, if y(t) > x(t), the above gradient would evaluate to **-a**, as **y(t)** is constant with respect to **x(t)**.

3. $$\frac{\partial(b \cdot |x(t) - x(t-1)|)}{\partial(x(t))}$$

   If **x(t) > x(t-1)**, the slope would be positive and the above equation evaluates to **b**, and if **x(t) < x(t-1)** then the slope would be negative and the above equation evaluates to **-b**. If **x(t) == x(t-1)**, then the curve would be flat and above equation evaluates to **0**.

In this way, the gradient of the function would be the sum of the above three terms. We have coded the above into **calculate_gradient()** function as below:

```
def calculate_gradient(x, y, t, p, a, b):
    gradient_first_term = p
    gradient_second_term = -a if y[t] > x[t] else 0
    if x[t] > x[t - 1]:
      gradient_third_term = b
    elif x[t] < x[t - 1]:
      gradient_third_term = -b
    else:
      gradient_third_term = 0


    gradient = gradient_first_term + gradient_second_term + gradient_third_term

    return gradient
```

We used the above equation to calculate the gradient at each step. Our algorithm is as follows:

```
def online_gradient_descent(y, learning_rate, p, a, b):
    n = len(y)
    x = [0.0] * n
    for t in range(n - 1):
        gradient = calculate_gradient(x, y, t, p, a, b)
        x[t + 1] = x[t] - learning_rate * gradient
    return x
```

We initially chose a learning rate as 0.02 to get an idea of whether the algorithm was working well. We got objective values in a range similar to the offline algorithm values.

```
[ ]  x = onlineGradientDescent_algo(home1.tolist(), learning_rate = 0.02)
```

```
[ ]  ogd_cost=cost(x, home1.tolist())
     print("\nThe objective value for Online Gradient Descent for Home1 is", ogd_cost)
```

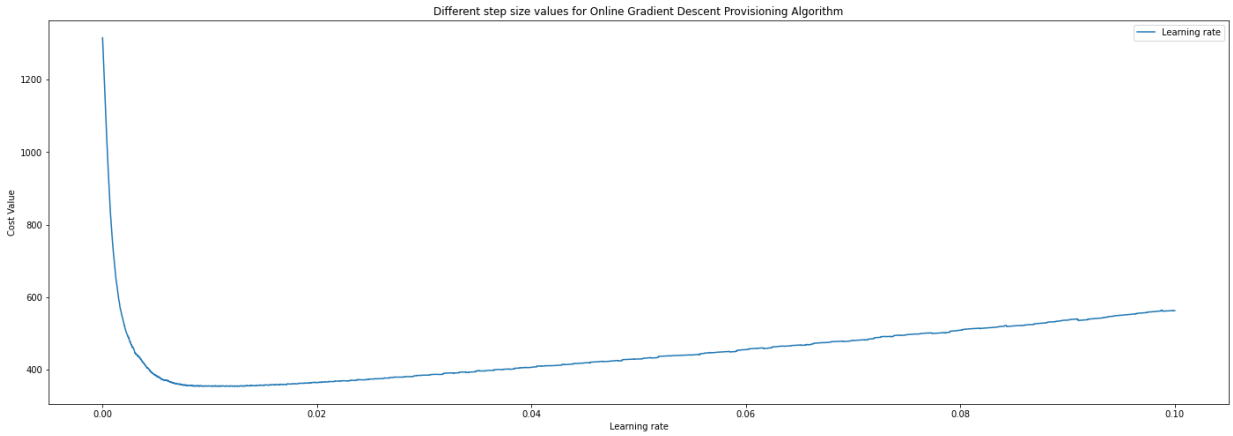
```
The objective value for Online Gradient Descent for Home1 is 364.8800533340003
```

This indicates that our algorithm is correct.

To find the optimal learning rate, we step through 10000 learning rates starting from 0.00001 with a step size increase of 0.00001. For each value of learning rate, we run the OGD algorithm. This gives us the best learning rate with precision of 5 decimals:

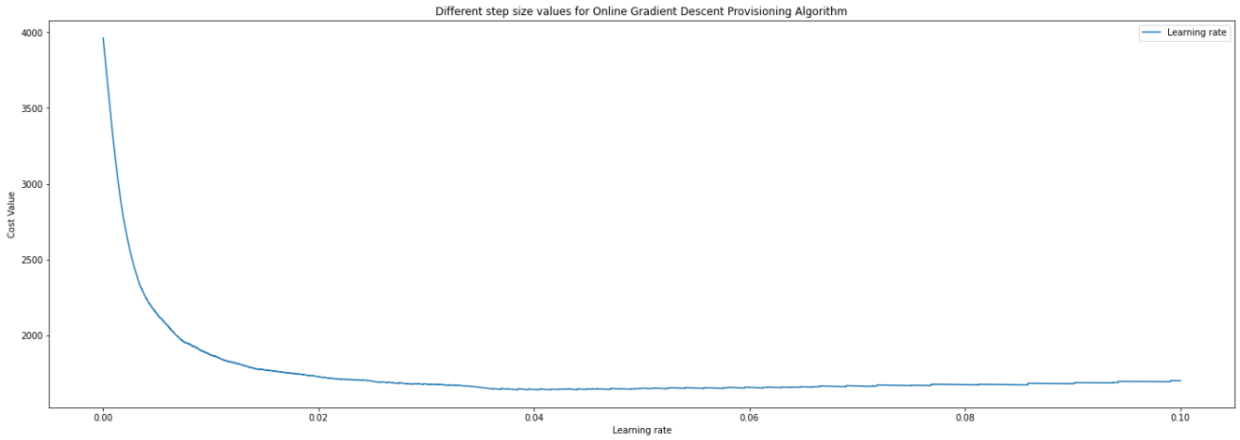
```
costs = []
learning_rates = []
lr = 0
for i in range(10000):
    lr += 0.00001
    x = onlineGradientDescent_algo(home1.tolist(), learning_rate = lr)
    learning_rates.append(lr)
    costs.append(cost(x ,home1.tolist()))
```

Here is a plot of OGD cost vs Learning rate for Home 1:



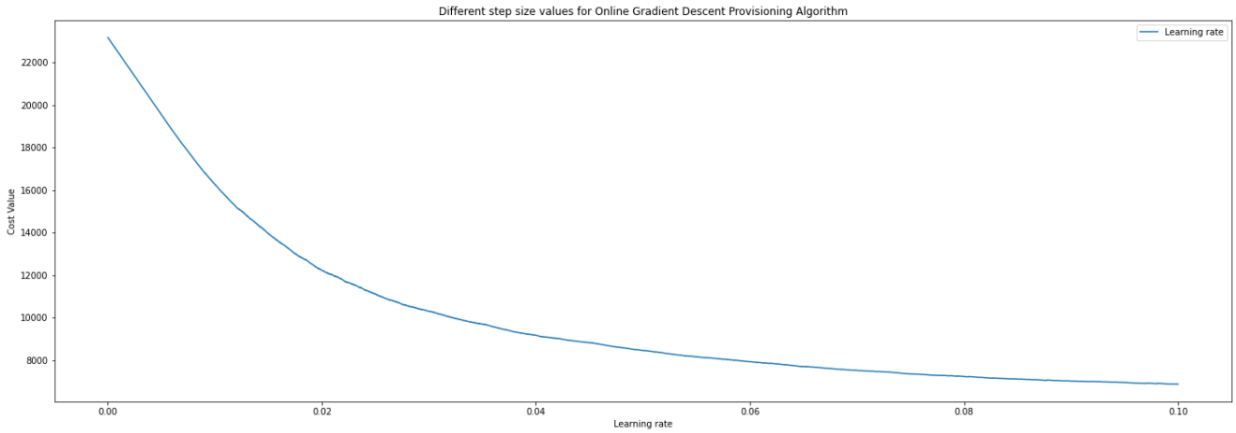

For Home 1, the learning rate corresponding to minimum cost is **0.01188**.

We repeat the same process for the other two homes as well to find the optimal learning rate for the Online Gradient Descent algorithm.
Here is a plot of OGD cost vs Learning rate for Home 2:




For Home 2, the learning rate corresponding to minimum cost is **0.03849**.

Here is a plot of OGD cost vs Learning rate for Home 3:

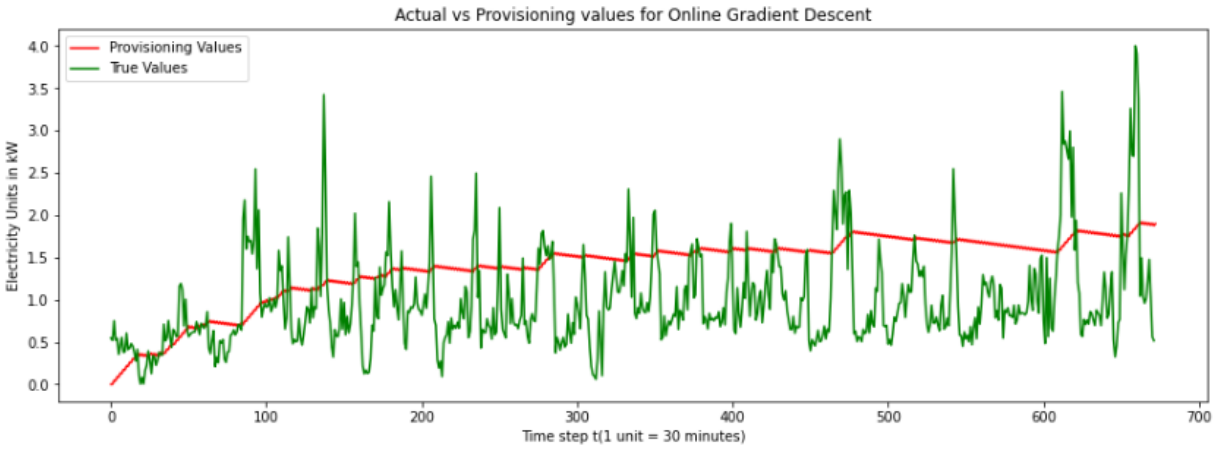



For Home 3, the learning rate corresponding to minimum cost is **0.099609**.

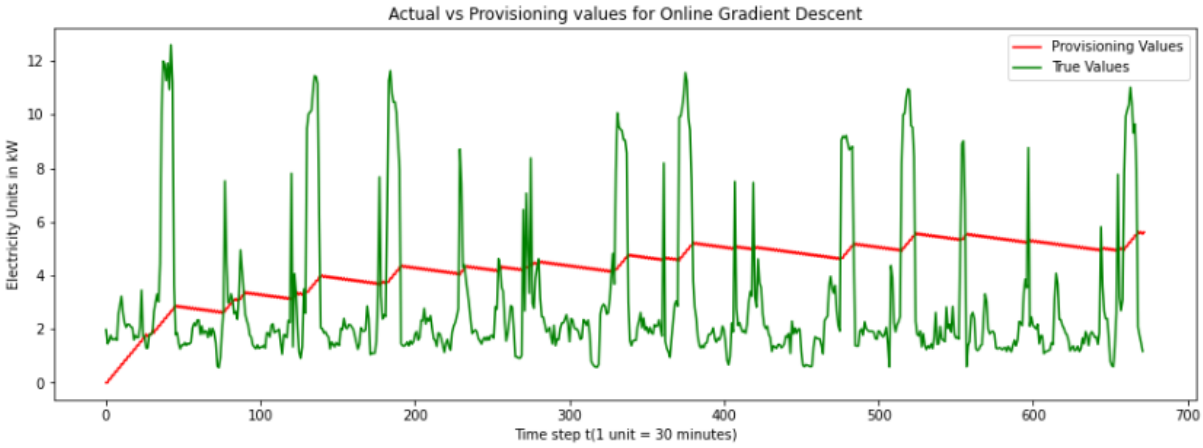
We get the following results for the three homes:

| | Optimal learning rate | OGD cost |
|---|---|---|
| Home 1 | 0.01188 | 353.85 |
| Home 2 | 0.03849 | 1639.23 |
| Home 3 | 0.09960 | 6871.17 |

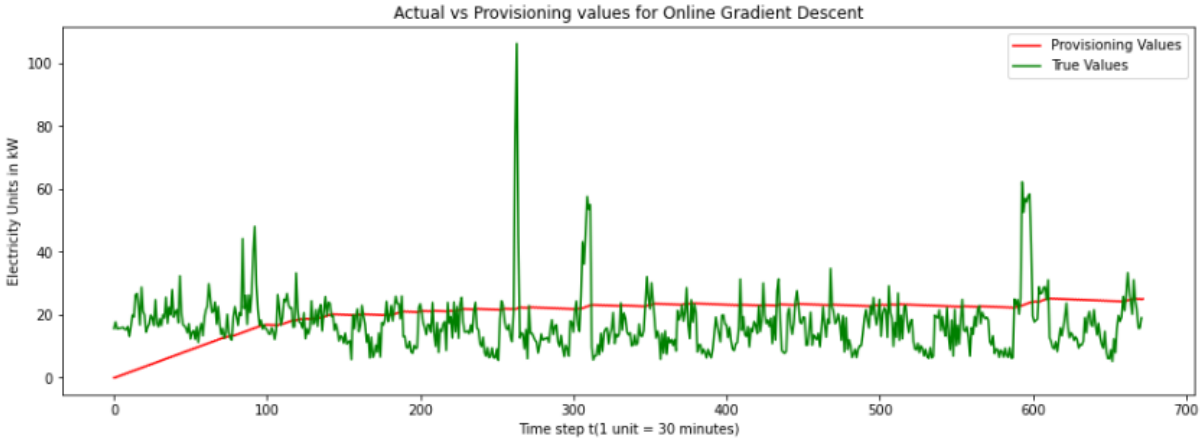
Home 1:




Home 2:




Home 3




# Receding Horizon Control

As we only had a superficial understanding of Receding Horizon Control and other Online Convex Optimization algorithms, we referred to various research papers to understand the concept better. Mainly, we understood the concept of RHC from the paper titled **'Using Predictions in Online Optimization: Looking Forward with an Eye on the Past'** by Chen et.al.

## Overview of RHC:

Receding Horizon Control (RHC) uses a prediction horizon / window of size **w**. At any given time **t**, RHC uses the predictions for the current horizon, determines the trajectory of '**w**' number of true values at time **t**: **x(t)** using an objective function defined as:

$$\sum_{t=1}^{T} p(t)x(t) + a * \max\{0, y(t) - x(t)\} + b|x(t) - x(t-1)|$$

As given in the problem statement, the number of time steps T = 672, a=b=$ 4 / kWh and p = $ 0.4 / kWh indicate the price of electricity used in terms of kWh. But since our data is bi-hourly, we need to consider only half the price of electricity hence the values a, b, are divided by 2 before being fed to the equation.

Then the algorithm minimizes the cost within that horizon/window, and then commits only the first true value from the horizon in that trajectory.

RHC is entirely "forward looking", which means it does not take into account the actions it's own in previous timesteps and thus incurs penalty in costs where the **consecutive true values are far apart** but also **enables quick response** to improved predictions. It obtains the optimal actions over the window (t + 1, t + w), at time **t**, given the starting state x(t) and a prediction window (horizon) of length w. It solves the convex optimization problem for each window as it slides over the time series.
For t=0 , we initialize the window with zero values as we do not have any values from previous steps.

- Finding the optimal window/horizon size:
  We first tried out random window sizes and obtained optimal values. But this does not give us a holistic idea of the impact of varying window sizes. Hence we finally started an incremental search for the window size that can yield optimal value with RHC.

  For example in Home 2:

1. $W \in [1,5]$ : 5 was the window size for optimal cost
2. $W \in [1,10]$ : 9 was the window size for optimal cost
3. $W \in [1,20]$ : 9 was still the best window size for optimal cost

   After this we concluded that increasing window size further beyond would not optimize the cost any further.

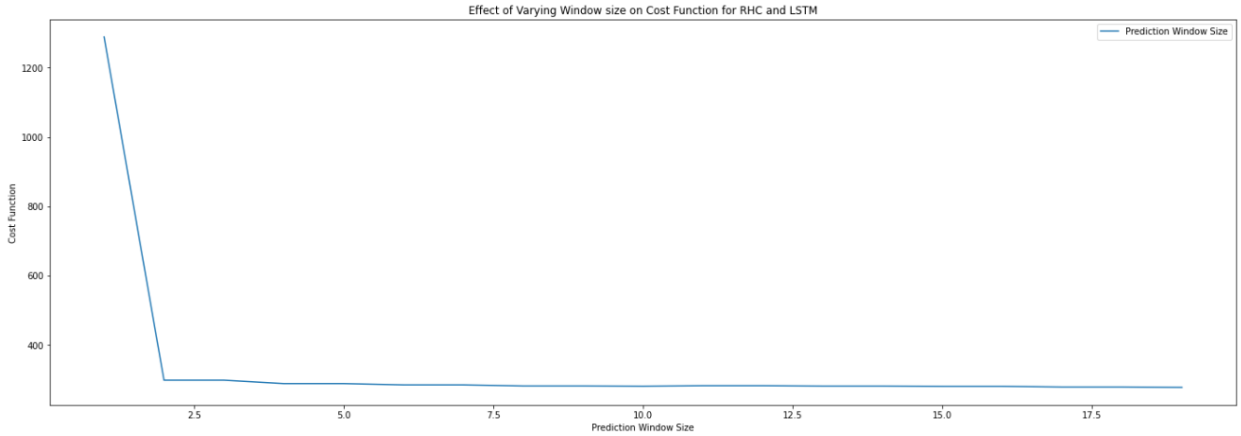
| | LSTM with RHC | ARIMA with RHC |
|---|---|---|
| Home1 | 19 | 19 |
| Home2 | 9 | 9 |
| Home3 | 9 | 9 |

Table: Optimal window sizes

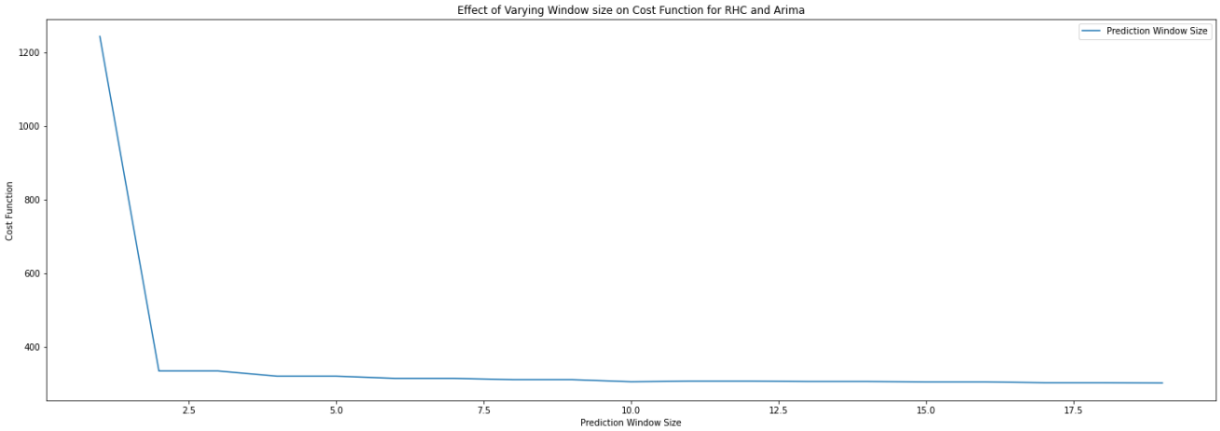
## Varying window sizes to see its effect on cost

1. Home 1:

RHC with LSTM

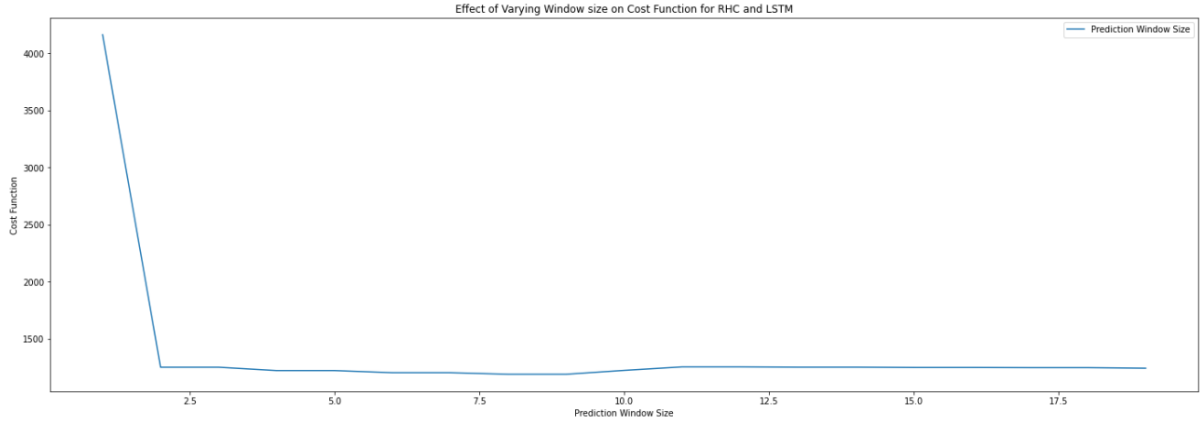



RHC with ARIMA

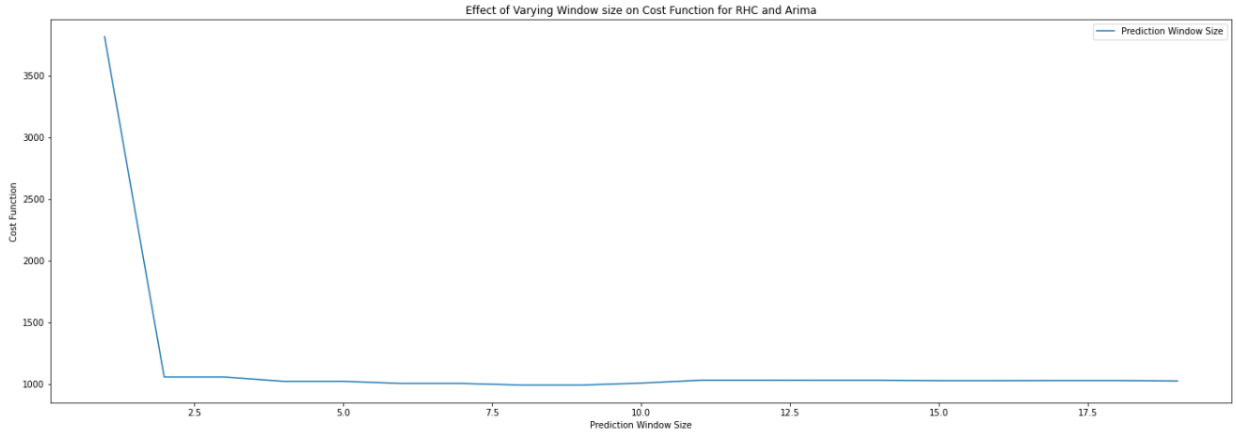



2. Home 2

RHC with LSTM

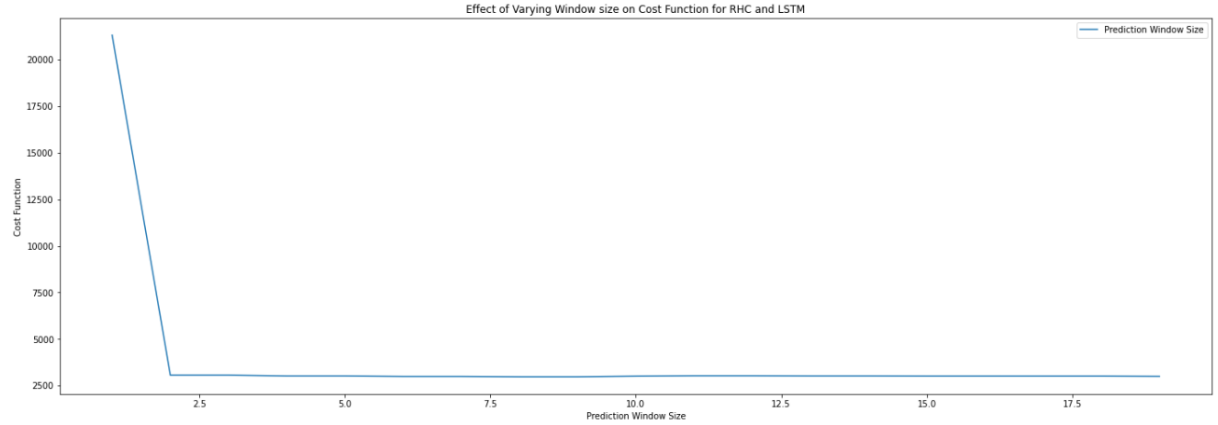



RHC with ARIMA

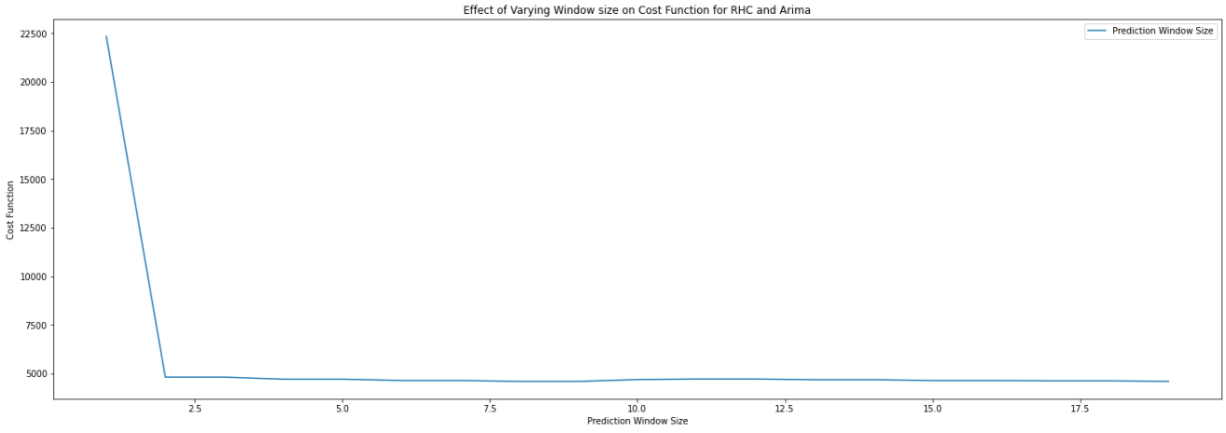



3. Home 3

RHC with LSTM




RHC with ARIMA




# Commitment Horizon Control

CHC is built on top of RHC but it allows commitment of a fixed number of steps(Let's say **c**). For the least value of commitment, **c** = 1, the algorithm behaves like RHC and for the

maximum value of commitment **c** = windowSize it behaves like the AFHC algorithm (mentioned in the paper). So, the CHC algorithm comprises the best properties of both algorithms and is expected to perform better than RHC. Let us find out if that is true.

In CHC, we use limited commitment where commitment level **c**
uses a prediction window of size w but then executes only the first **c** ∈ [1, w] actions which can be visualized by Figure 2.
[Reference : Using Predictions in Online Optimization: Looking Forward with an Eye on the Past]

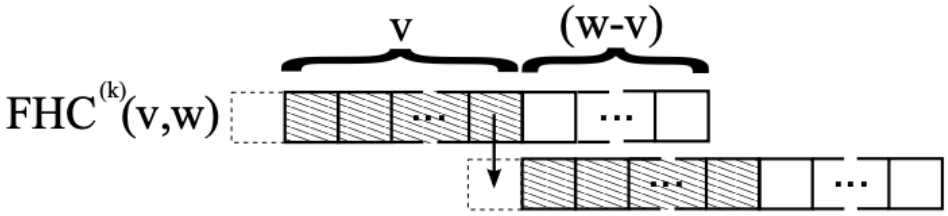



**Figure 2: Fixed Horizon Control with commitment level $v$: optimizes once every $v$ timesteps for the next $w$ timesteps and commits to use the first $v$ of them.**

CHC(v, w), the CHC algorithm with prediction window w and commitment level v, averages over v FHC algorithms with prediction window w and commitment level v

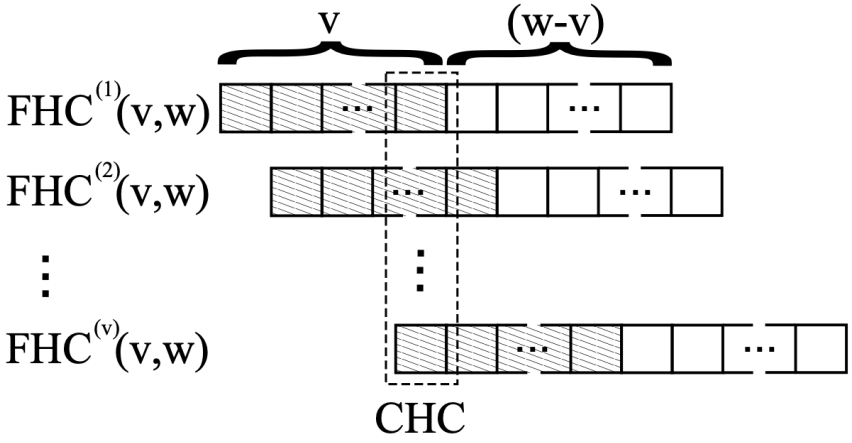



**Figure 3: Committed Horizon Control: at each timestep, it averages over all $v$ actions defined by the $v$ FHC algorithms with limited commitment.**

[Reference : Using Predictions in Online Optimization: Looking Forward with an Eye on the Past]

Note:

1. We observed that the number of iterations for commitment size **c** should be **T + commitment size - 1**.

2. Can window size be equal to commitment size? Yes. But it cannot be less than commitment. The reason is that we need to take the mean of values over a window with length = commitment-level.

- Finding the optimal commitment/horizon level:
  We first tried out random commitment levels and obtained optimal values. But this does not give us a holistic idea of the impact of varying levels. Hence we finally started an incremental search for the horizon size that can yield optimal value with CHC.

  For example in Home 1:
  1. $C \in [1,5]$ : 5 was the horizon size for optimal cost
  2. $C \in [1,10]$ : 10 was the horizon size for optimal cost
  3. $C \in [1,20]$ : 19 was still the best horizon size for optimal cost

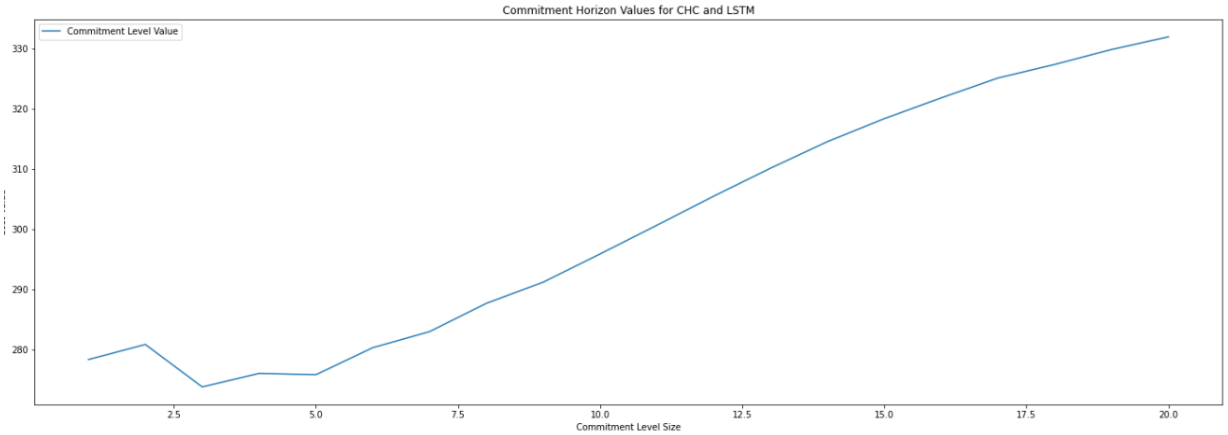
  After this we concluded that increasing the commitment horizon further beyond would not optimize the cost any further.

| | LSTM with CHC | ARIMA with CHC |
|---|---|---|
| Home1 | 3 | 5 |
| Home2 | 3 | 3 |
| Home3 | 1 | 1 |

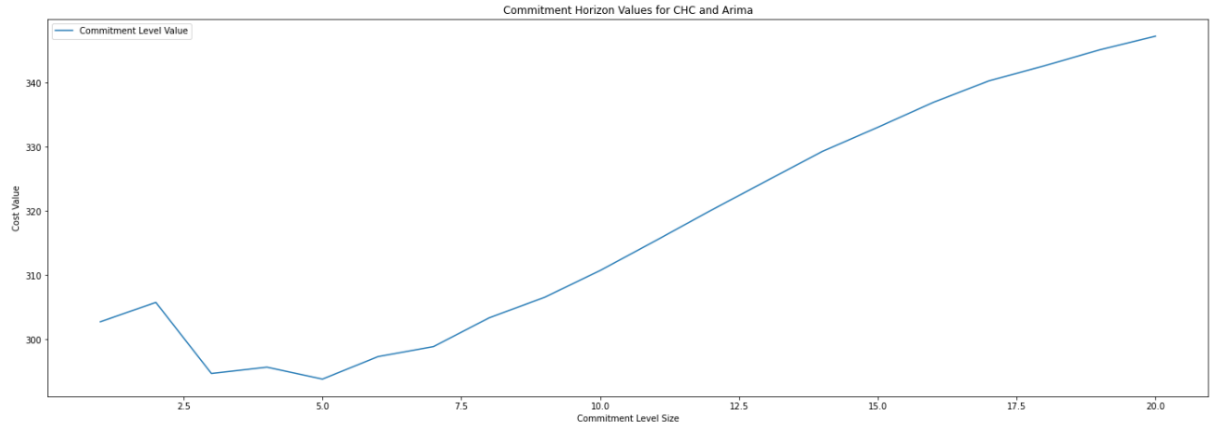
## Varying commitment level to see its effect on cost

### Home 1

CHC with LSTM

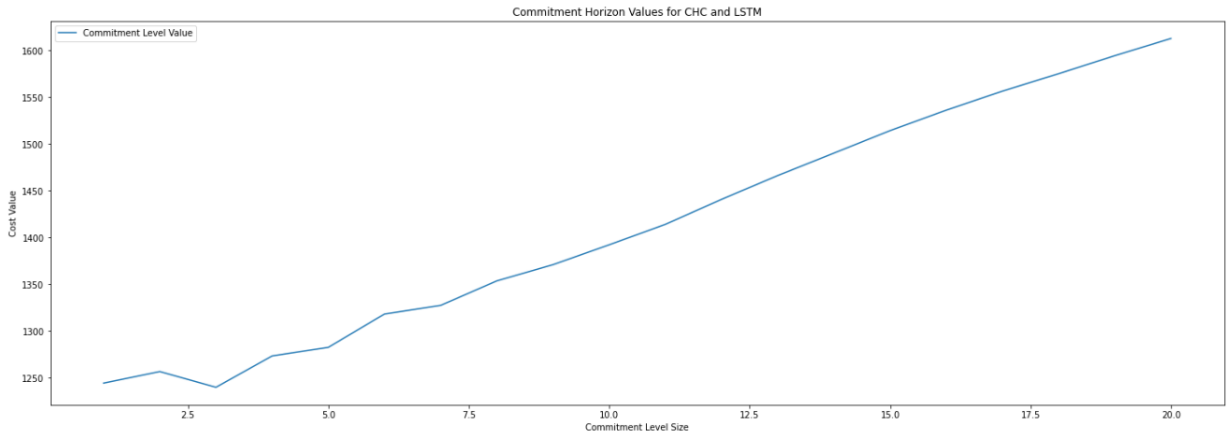



CHC with ARIMA

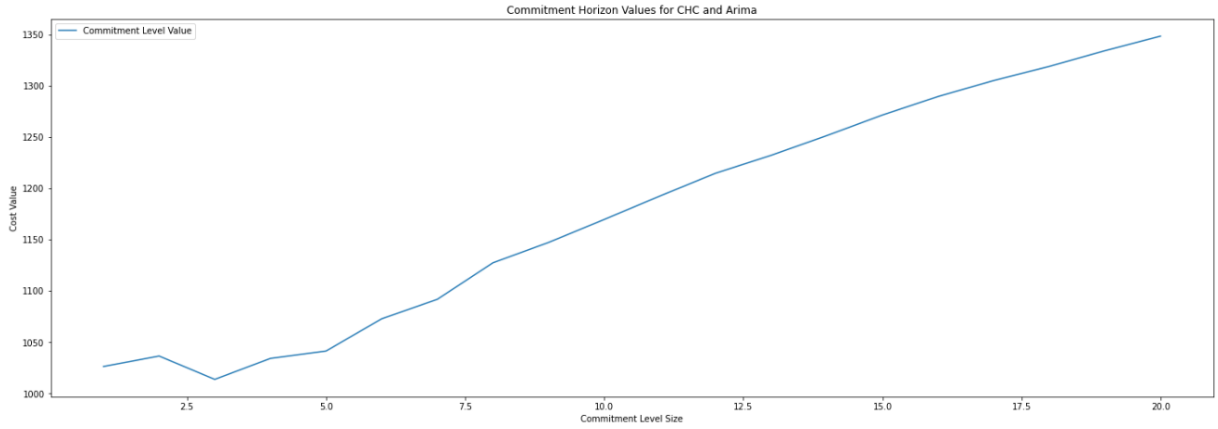



**Home 2**:

CHC with LSTM

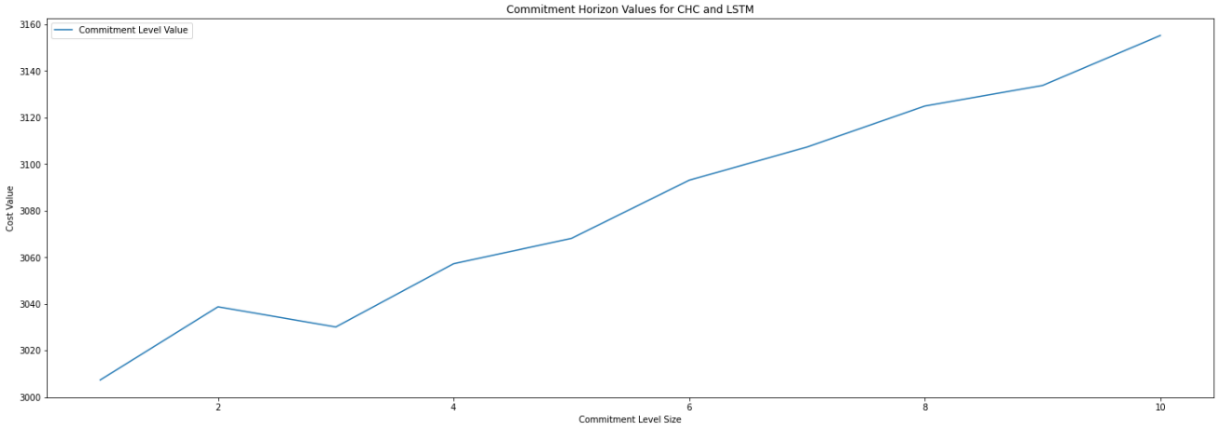



CHC with ARIMA

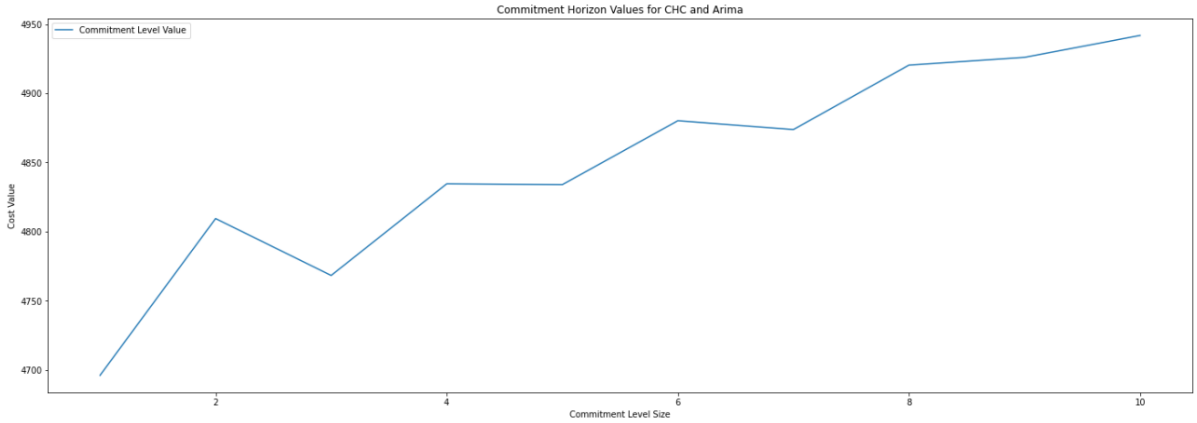



## Home 3:

CHC with LSTM




CHC with ARIMA



# Varying ‘a’ and ‘b’ with best optimization algorithm and provisioning algorithm

We varied both a and b for CHC and LSTM combination as it was the best combination of predictions algorithm and provisioning algorithm.
Window size and horizon size were used from the previous observations which yielded optimal values of the cost function.
When we varied a we kept b constant and when b was varied, a was constant.

The below graphs were obtained from this process for each Home data.

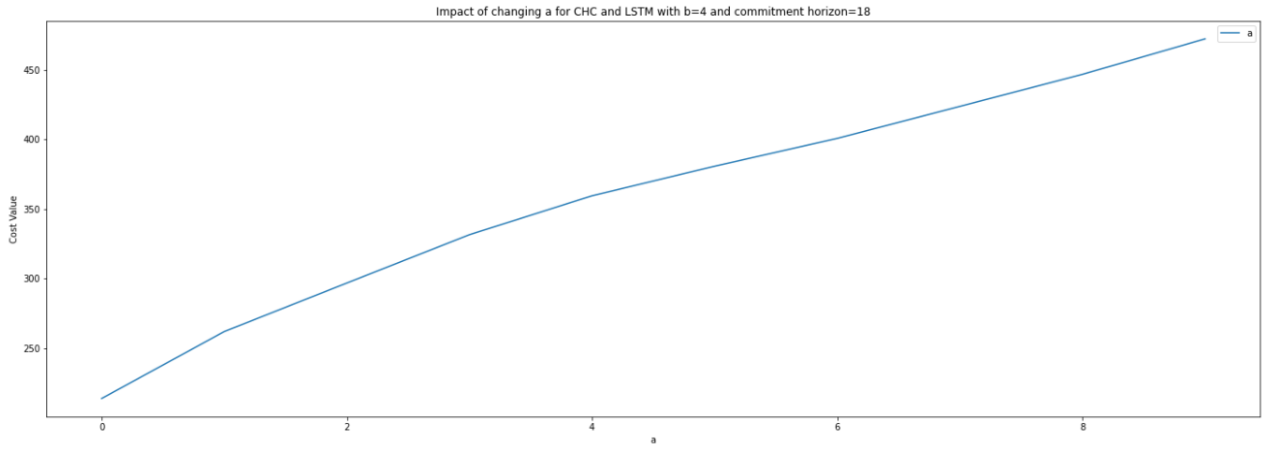
For all the homes, as we increased both a and b,we had the following common observations:

1. For constant b , increasing a: **for linear increments in a value, the cost function increased with nearly linear growth.**
2. For constant a , increasing b: **for linear increments in b value, the cost function increased with ‘a’ in a jittered fashion (increasing growth with jitter)**

## Home 1:

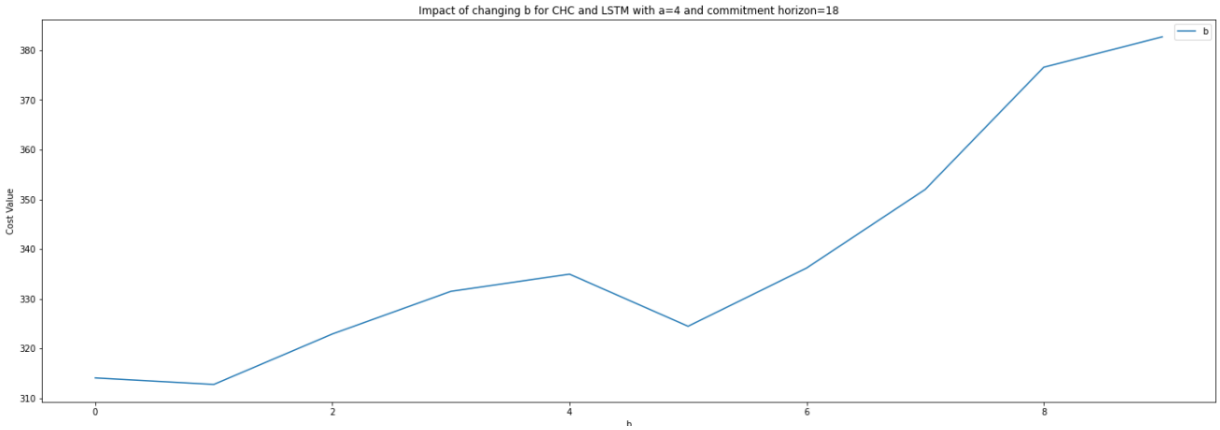
Varying **a** keeping **b** constant:

**Impact of changing a for CHC and LSTM with b=4 and commitment horizon=18**



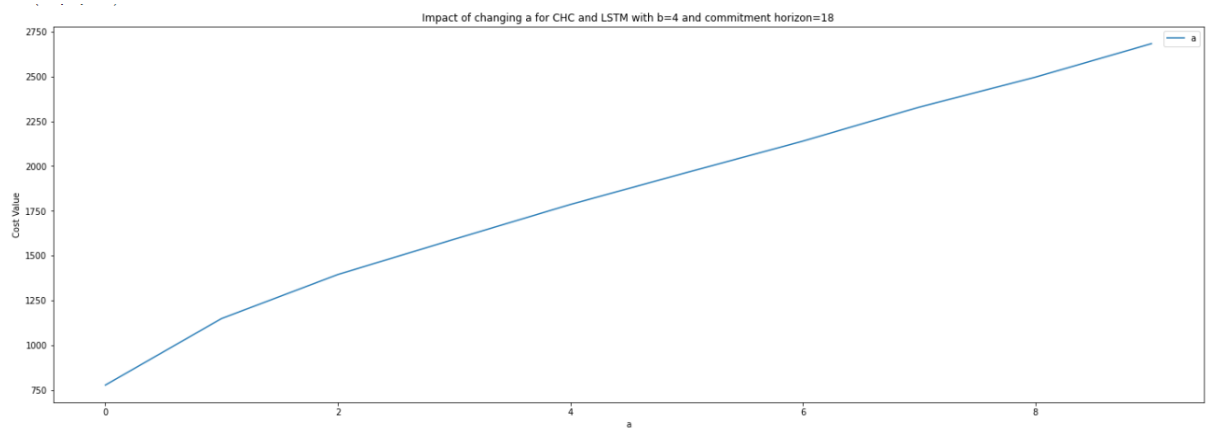

Varying **b** keeping **a** constant:

**Impact of changing b for CHC and LSTM with a=4 and commitment horizon=18**

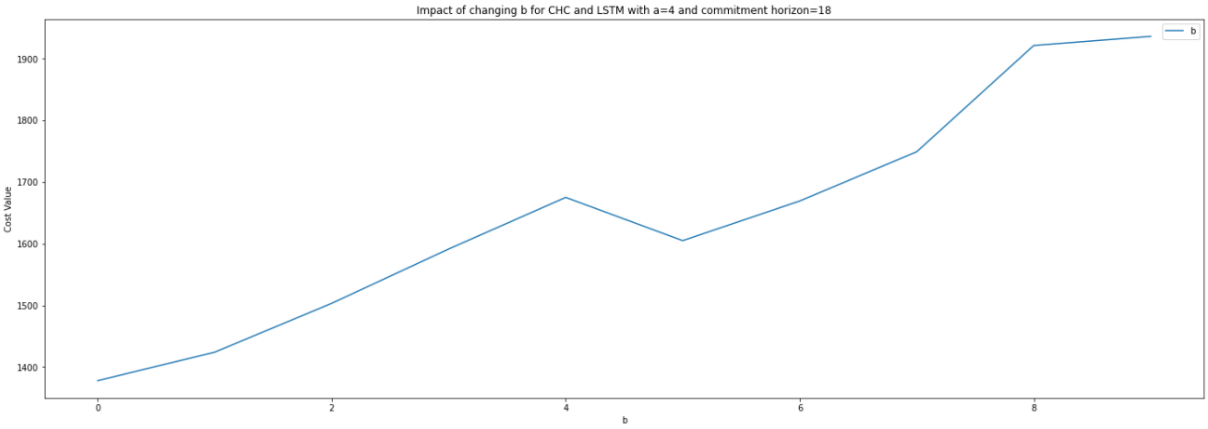



## Home 2:

Varying **a** keeping **b** constant:

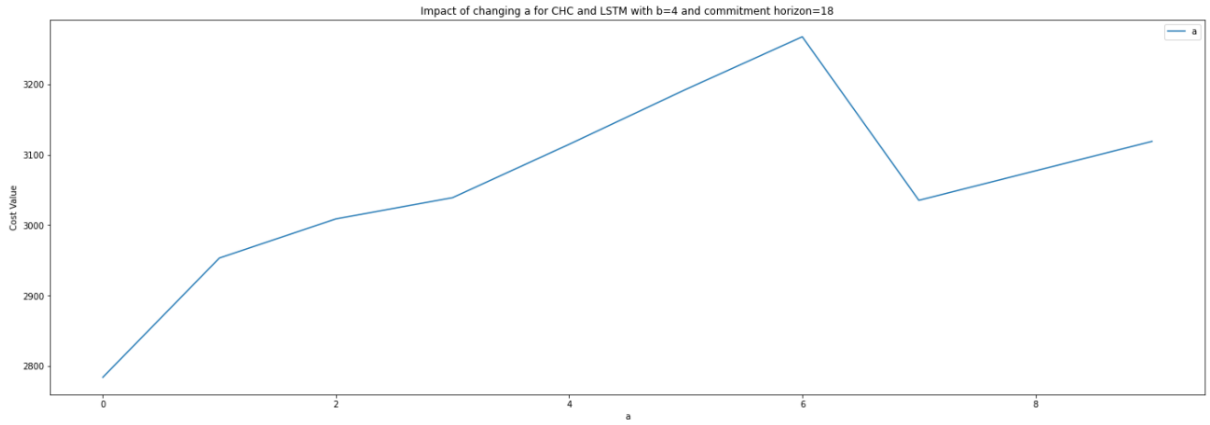



Varying **b** keeping **a** constant:

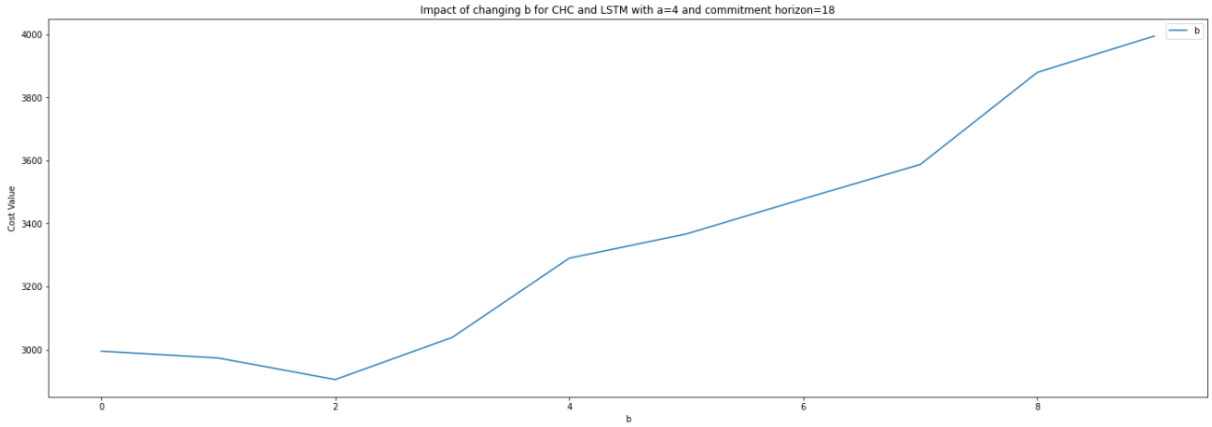



## Home 3:

Varying **a** keeping **b** constant:



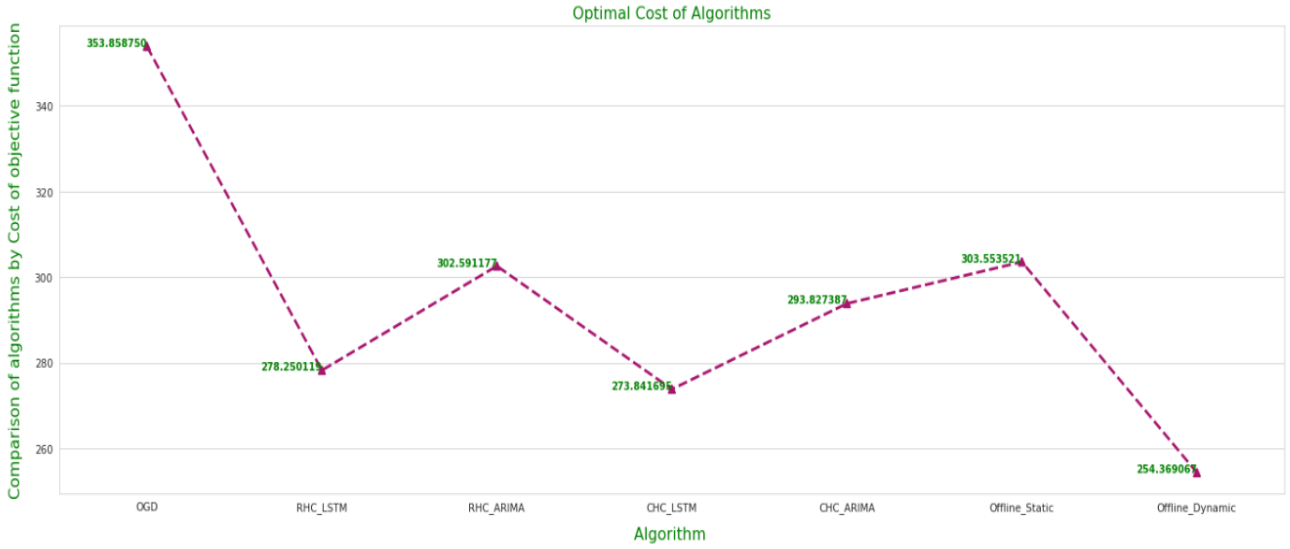

Varying **b** keeping **a** constant:



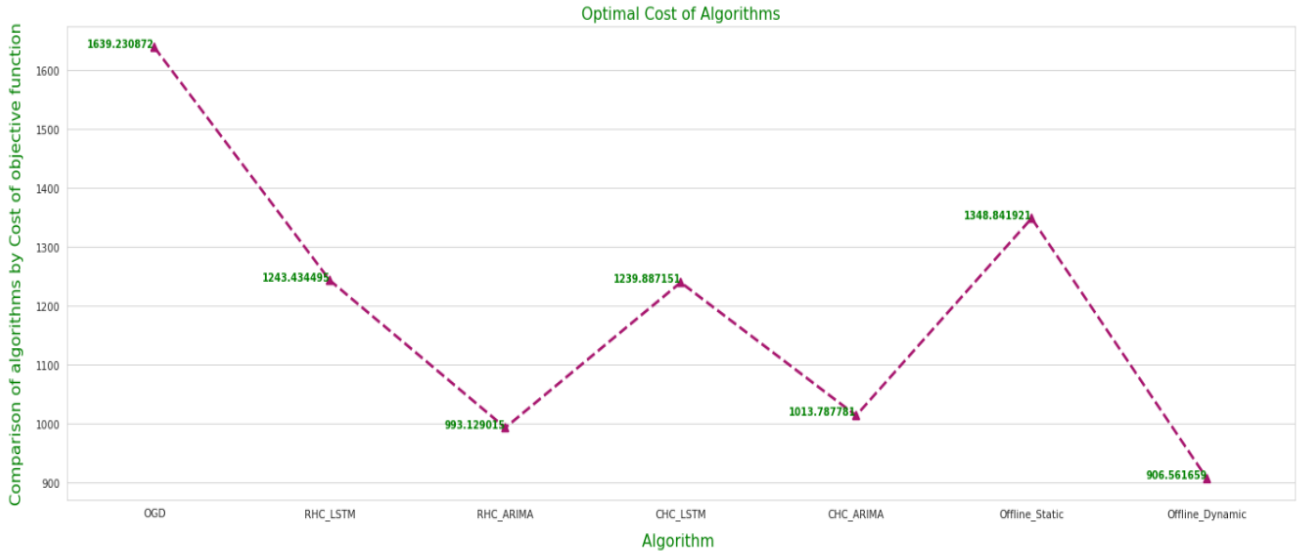

## Comparison of the objective function values for Offline Static and Dynamic, OGD, RHC with ARIMA and LSTM, CHC with ARIMA and LSTM

We are comparing costs of our algorithms to those of offline static and dynamic solutions.

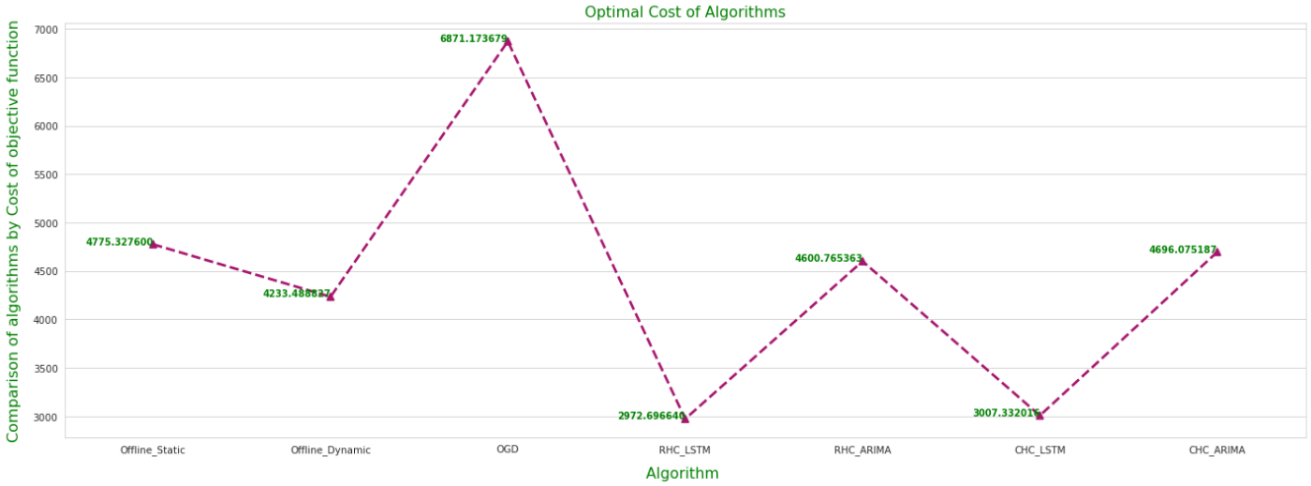
Home 1:


Optimal Cost of Algorithms
Comparison of algorithms by Cost of objective function
353.858750
278.250115
302.591171
273.841695
293.827387
303.553521
254.369067
340
320
300
280
260
OGD
RHC_LSTM
RHC_ARIMA
CHC_LSTM
CHC_ARIMA
Offline_Static
Offline_Dynamic
Algorithm

Home 2:

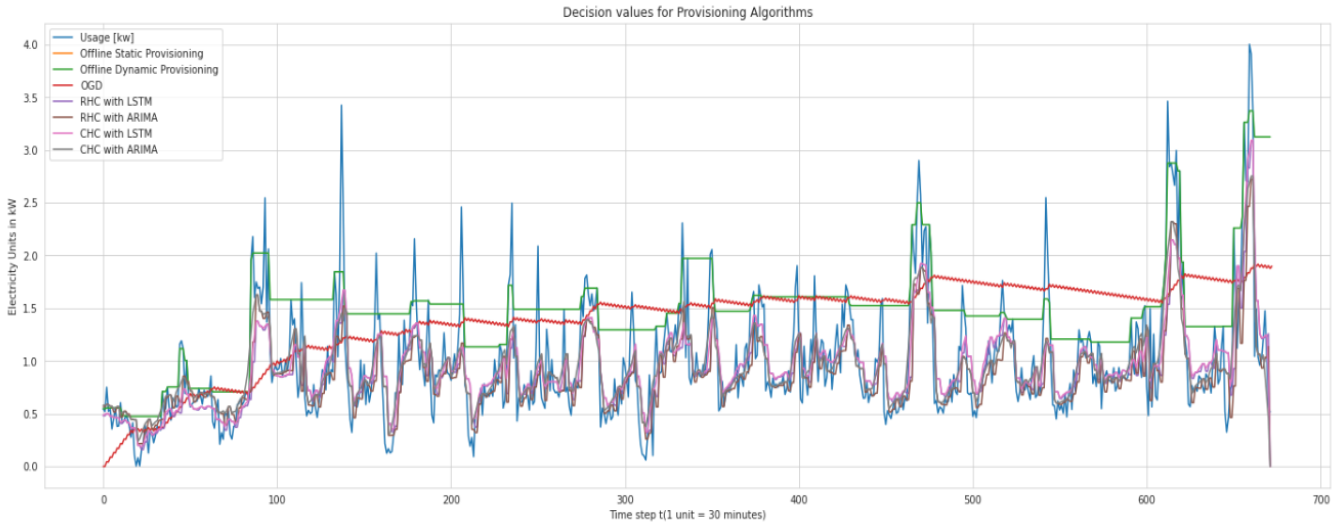

Optimal Cost of Algorithms
Comparison of algorithms by Cost of objective function
1639.230872
1243.434495
993.129075
1239.887151
1013.787781
1348.841921
906.561655
1600
1500
1400
1300
1200
1100
1000
900
OGD
RHC_LSTM
RHC_ARIMA
CHC_LSTM
CHC_ARIMA
Offline_Static
Offline_Dynamic
Algorithm

Home 3:

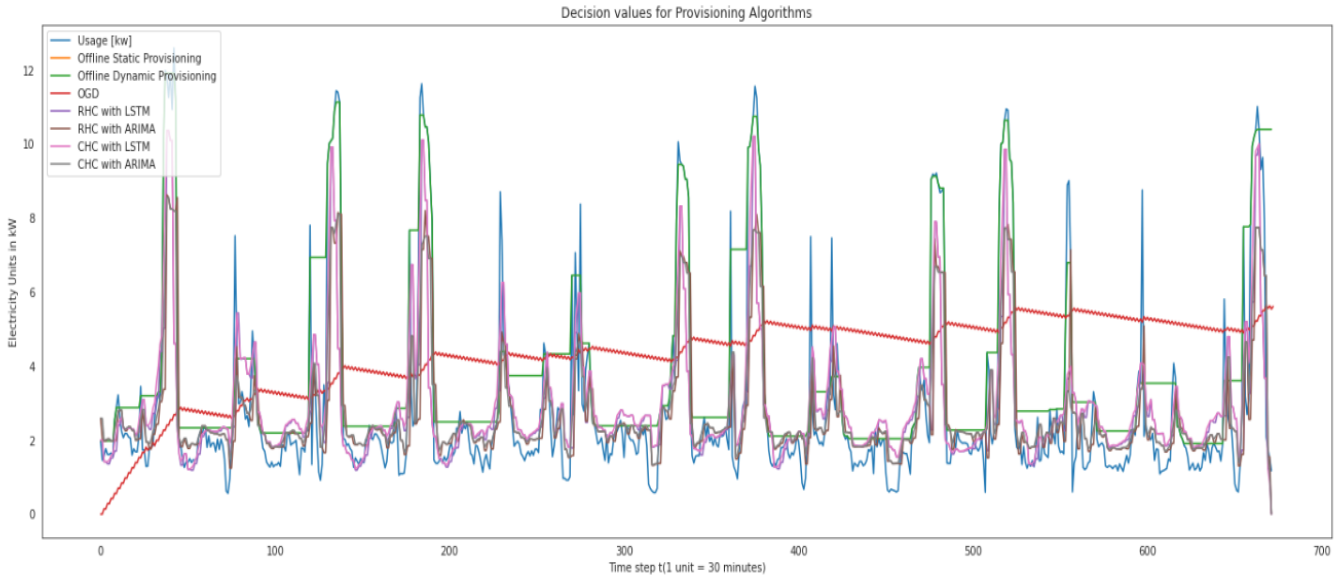



## Comparison of electricity provision values for Offline Static and Dynamic, OGD, RHC with ARIMA and LSTM, CHC with ARIMA and LSTM:

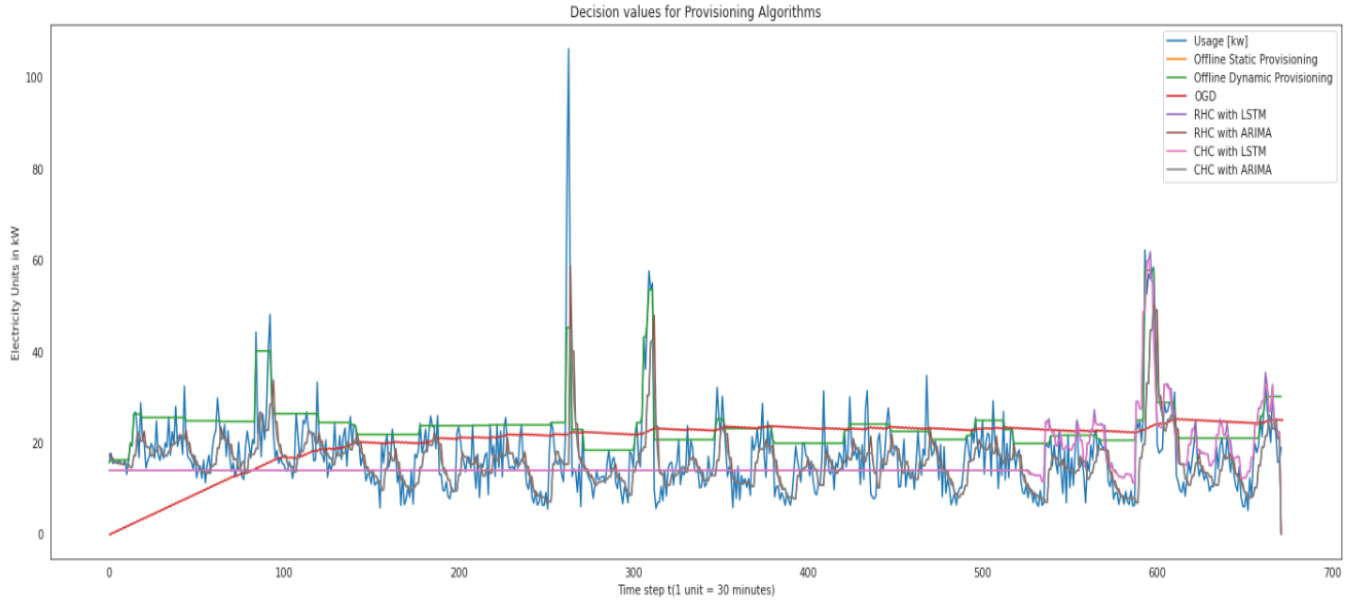
Home 1:



## Home 2:


Decision values for Provisioning Algorithms
Usage [kw]
Offline Static Provisioning
Offline Dynamic Provisioning
OGD
RHC with LSTM
RHC with ARIMA
CHC with LSTM
CHC with ARIMA
Electricity Units in kW
Time step t(1 unit = 30 minutes)

## Home 3:


Decision values for Provisioning Algorithms
Usage [kw]
Offline Static Provisioning
Offline Dynamic Provisioning
OGD
RHC with LSTM
RHC with ARIMA
CHC with LSTM
CHC with ARIMA
Electricity Units in kW
Time step t(1 unit = 30 minutes)

A summary of optimal costs for all the algorithms is as follows:

| Data | Offline Static | Offline Dynamic | Online Gradient Descent | RHC (LSTM) | RHC (ARIMA) | CHC (LSTM) | CHC (ARIMA) |
|---|---|---|---|---|---|---|---|
| Home1 | 303.55 | 254.369 | 353.858 | 278.25 | 302.59 | 273.8416 | 293.827 |
| Home2 | 1348.84 | 906.56 | 1639.23 | 1243.43 | 993.12 | 1239.88 | 1013.78 |
| Home3 | 4775.32 | *4233.48* | 6871.17 | *2972.69* | *4600.76* | *3007.33* | *4696.07* |

*Best performing offline algorithms highlighted in yellow*

*Best performing online algorithms highlighted in green*

We see that the offline dynamic algorithm is the algorithm with the minimum cost for almost all the cases. This is expected as the offline dynamic algorithm has access to the entire data at once and can fully optimize provisioning values to obtain minimum cost. (This trend is not followed for Home 3).

Considering online algorithms, the horizon algorithms consistently beat OGD. CHC and RHC have great performance across the board for combinations of both prediction algorithms. However, we see that CHC with LSTM seems to be marginally better in the average case.

Regret as a metric:

Regret is another metric which is used to quantify the performance of an algorithm, but with respect to an offline algorithm. We can have regret with respect to the static offline algorithm, termed as ‘static regret’. And we can have regret with respect to the dynamic offline algorithm, termed as ‘dynamic regret’.

They are calculated as below:

**Static Regret = Algorithm cost - Static Offline Cost**

**Dynamic Regret = Algorithm cost - Dynamic Offline Cost**

The term ‘regret’ can be explained via the following idea: if an online algorithm which can choose from a set of actions chooses an action which gives some cost **c**, and upon selecting, the algorithm is shown an action it could have taken instead, corresponding to a lower cost **c’**. Then the algorithm experiences an instantaneous **regret = c - c’**.

We also present our results using the regret metric:

| Data | OGD Static Regret | OGD Dynamic Regret | RHC Static Regret (LSTM) | RHC Static Regret (ARIMA) | RHC Dynamic Regret (LSTM) | RHC Dynamic Regret (ARIMA) | CHC Static Regret (LSTM) | CHC Static Regret (ARIMA) | CHC Dynamic Regret (LSTM) | CHC Dynamic Regret (ARIMA) |
|---|---|---|---|---|---|---|---|---|---|---|
| Home 1 | 50.30 | 99.489 | -25.30 | -0.962 | 23.88 | 48.22 | -29.711 | -9.726 | 19.47 | 39.458 |
| Home 2 | 290 | 732 | -105 | -355 | 336 | 86.56 | -108 | -335 | 333 | 107 |
| Home 3 | 2095.8 | 2637.68 | -1802.63 | -174.56 | -1260.79 | 367.276 | -1767.99 | -79.252 | -1226.15 | 462.58 |

## Deterministic Algorithm selection

For the deterministic algorithm selection, we are choosing a history of 16 time intervals (8 hours). We will be using the algorithm cost (using the same objective function we have been using so far) as a comparison metric.

Our algorithm selection works as follows:

1. Look back at a history of 16 time intervals
2. Find out the best performing algorithm in this horizon, i.e. algorithm with minimum cost
3. Use algorithm found in the previous step to predict for the next 16 time intervals, i.e. 8 hours
4. Continue until all predictions have been made.

Using the above algorithm, we will have 672 / 16 = 42 ‘rounds’ where the three algorithms will compete against each other.

We ran the algorithm selection for all 3 homes, but will be illustrating the process using Home 2 as an example.

These are the results of running the deterministic algorithm for OGD, RHC and CHC:

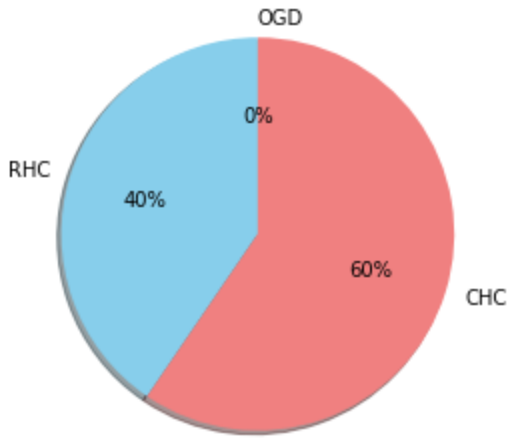



Figure: Percentage of wins of OGD, RHC and CHC in deterministic algorithm selection

To verify the deterministic nature of the algorithm, we ran the selection match in a loop to confirm if the results are deterministic in nature. We wanted to run more matches, but as running a single match takes more than a minute, we ran 5 matches in succession:

```
for i in range(5):
  x, algorithms_score = deterministic_selection(home2.tolist(), home2LSTM['forecast'].tolist(), 16)
  print("Match #" + str(i + 1) + ": Algorithms score: " + str(algorithms_score))
```

```
Match #1: Algorithms score: {'OGD': 0, 'RHC': 17, 'CHC': 25}
Match #2: Algorithms score: {'OGD': 0, 'RHC': 17, 'CHC': 25}
Match #3: Algorithms score: {'OGD': 0, 'RHC': 17, 'CHC': 25}
Match #4: Algorithms score: {'OGD': 0, 'RHC': 17, 'CHC': 25}
Match #5: Algorithms score: {'OGD': 0, 'RHC': 17, 'CHC': 25}
```

We see that each match plays out exactly with the same results. This confirms our expectations about the deterministic algorithm selection.

## Observations about deterministic algorithm selection:

According to our results in earlier sections of the assignment, we know that OGD performs fairly worse compared to CHC and RHC algorithms. We observe that the deterministic selection method is very strict in the sense that it will never allow a worse performing algorithm to win a round. This is the reason that we see that OGD's score is 0.

**Note**: *For algorithm selection, we have used y_hat (i.e. prediction values) for all three algorithms: OGD, RHC and CHC. We understand that OGD is supposed to use true values, however, in a competitive environment of algorithm selection, feeding different data to algorithms very clearly affects the metric, and as a result leads to the incorrect algorithm being chosen as the*

*winner. We had tried using true values for OGD before, and that made OGD win **90%** of the rounds, which was entirely contradictory with our prior results. We tracked down the cause of the difference in data for the algorithms. Once we fed the same data (predictions) to all three algorithms, we got results that made sense (see below).*

## Randomized Algorithm selection

The overall structure of the randomized algorithm selection is similar to the deterministic algorithm selection. However, the main difference is in the way the algorithm is actually selected. In a deterministic selection, the best performing algorithm is chosen directly. However, in randomized selection, there is a stochastic component to the algorithm selection. We initially assign each algorithm equal weights. After one round in the match, we compare the performance of the algorithms and update the weights in the following way: winner gets a bonus and loser gets a penalty.

We formalize this into an algorithm as follows:

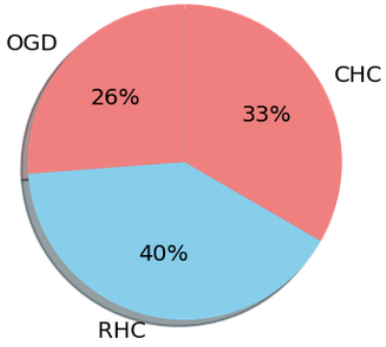
1. Assign each algorithm equal weights.
2. Decide on bonus/penalty values
3. Look back at a history of 16 time intervals
4. Rank the algorithms according to their performance in this horizon
5. Update the weights of the algorithms according to the rank
6. Choose an algorithm randomly using the updated weights
7. Use algorithm found in the previous step to predict for the next 16 time intervals, i.e. 8 hours
8. Run steps 3 through 8 until all predictions have been made.

We ran the randomized algorithm selection multiple times to see the different results. We used bonus = penalty = 0.7% (we chose 0.7% as we do not want any of the weights to go negative). As we have three algorithms, we initialize the weights to ⅓ for each of the 3 algorithms.

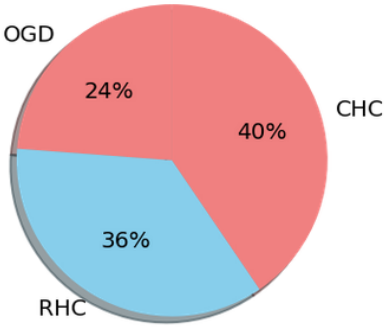
We ran the algorithm selection for all 3 homes, but will be illustrating the process using Home 2 as an example.

As running randomized selection once won't give enough insights, we ran the algorithm multiple times from the get go. Results are as follows:

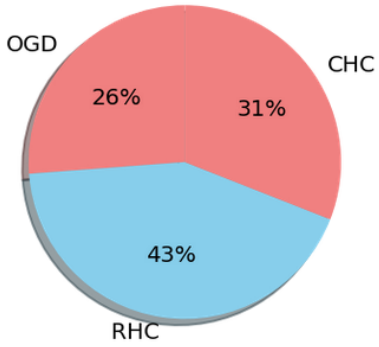
| Match # | Algorithms score | Algorithms weight |
|---|---|---|
| 1 | {'OGD': 11, 'RHC': 17, 'CHC': 14} | {'OGD': 0.0393, 'RHC': 0.4523, 'CHC': 0.5083} |
| 2 | {'OGD': 10, 'RHC': 15, 'CHC': 17} | {'OGD': 0.0393, 'RHC': 0.4523, 'CHC': 0.5083} |
| 3 | {'OGD': 11, 'RHC': 18, 'CHC': 13} | {'OGD': 0.0393, 'RHC': 0.4523, 'CHC': 0.5083} |
| 4 | {'OGD': 6, 'RHC': 16, 'CHC': 20} | {'OGD': 0.0393, 'RHC': 0.4523, 'CHC': 0.5083} |
| 5 | {'OGD': 5, 'RHC': 15, 'CHC': 22} | {'OGD': 0.0393, 'RHC': 0.4523, 'CHC': 0.5083} |



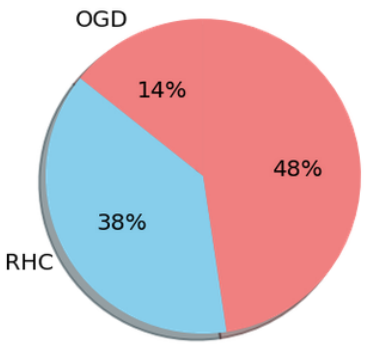

Match #1



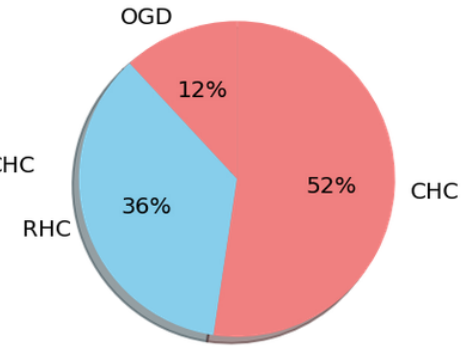

Match #2




Match #3




Match #4




Match #5

We see that CHC wins in a majority of the matches, except Match #3 where RHC is the winner. This is in line with our previous observations.

## Observations and insights about randomized algorithm selection:

- We see that randomized algorithm selection is not harsh as deterministic algorithm selection. The worst performing algorithm OGD is selected in a handful of rounds. We can also fine-tune this aspect of the algorithm by adjusting the penalty.
- Another important observation to make is that the resultant weights at the end of the algorithm are exactly the same in each match. This is because the actual performance of the algorithm is not stochastic in nature, thus the weight updates will always follow the same path across runs under the same conditions.
- There are a lot of changes we can make to the randomized selection by using different parameters, i.e. by changing the bonus and penalty, we can change how quickly the best algorithm starts dominating the stochastic selection. Apart from this, we can also make the weight updates asymmetric: the bonus could be more than the penalty or vice-versa. This changes the dynamic  This offers a great deal of flexibility and it offers a very tangible

# Contribution and work distribution

| **Name** | **Contribution percentage** | **Areas worked on** |
|---|---|---|
| Akanksha Kale (SBU ID: 113788594) | 33.33% | Commitment Horizon Control, Receding Horizon Control, Effect of Varying a and b, Data Preprocessing |
| Akhila Juturu (SBU ID: 114777498) | 33.33% | Data massaging, Online Gradient Descent, Comparison of Algorithms, Commitment Horizon Control algorithm |
| Mandar Laxmikant Mahajan (SBU ID: 113276053) | 33.33% | Offline static optimization, offline dynamic optimization, deterministic and randomized algorithm selection |

# References


- “Online Convex Optimization: Algorithms, Learning, and Duality” by Victor Sanches Portella [https://www.cs.ubc.ca/~victorsp/oco/thesis.pdf]
- ‘Using Predictions in Online Optimization: Looking Forward with an Eye on the Past’ by Chen et.al [https://niangjunchen.github.io/papers/OCO_prediction.pdf]