Interim Internship Report

(Start date: 28 May)

by

Saurabh Roychowdhury

&

Dinesh Mohanty

Indian Institute of Technology Bhubaneswar(**IIT**)

Bhubaneswar

India

Under the guidance of

Prof. Dr. Wessam Ajib

Deputy Director of Research

&

Prof. Dr. Elmahdi Driouch

Computer Sciences Department

L'Université du Québec à Montréal (**UQAM**)

Montréal

Canada

Content

Problem Statements:

Problem Statement 1:

Aim: Our first objective was to find the optimum set of power values to get maximum spectral efficiency as output.
Constraint: Pi should be less than or equal to a given max power per ue. For testing purpose, we have used 3 levels of power, i.e 1, 10, 100 mW, so that we can keep the run time of the testing process under a manageable time. For greater accuracy, further power levels can be incorporated in the algorithm without much worry about the time complexity as the reinforcement learning code takes negligible time and is scalable whereas the simulation time takes a major chuck of the run time which fortunately won't be contributing when we are deploying the RL model on real world environment as real time data is spontaneous.
Objective: Distribute total power among all UEs such that the cumulative distributive function of spectral efficiency plotted with respect to UEs is shifted towards the right as much as possible. Or in other words try to maximise the spectral efficiency for each UE.
Components of the reinforcement learning algorithm:
1) **Agent:** Central Server
2) **Action:** Distribute the power to all ues in an efficient manner. (pi)
3) **Reward:** area above the curve for attaining the best performance/ total power
4) **State:** P matrix

Problem Statement 2:

Aim: Computing D (UE-AP association matrix) vector using reinforcement learning.
Constraint: The constraints are almost the same as that of problem statement 1 except that here as an added constraint, each ue should get at least one ap. D(i,k) =1 for at least one ith ap for a given kth ue.
Objective: To shift the cumulative distribution function of spectral efficiency plotted with respect to UEs is towards the right as much as possible. Or in other words, maximise the spectral efficiency of the system.
But here the optimization parameter is different, here it will be an association matrix instead of power matrix.
Components of the reinforcement learning algorithm:
1) **Agent:** Central Server
2) **Action:** Change parameters of D. Each step corresponds to change in the association of ue and ap.
3) **Reward:** area above the curve for attaining the best performance.
4) **State:** D matrix

1)Results

Plots

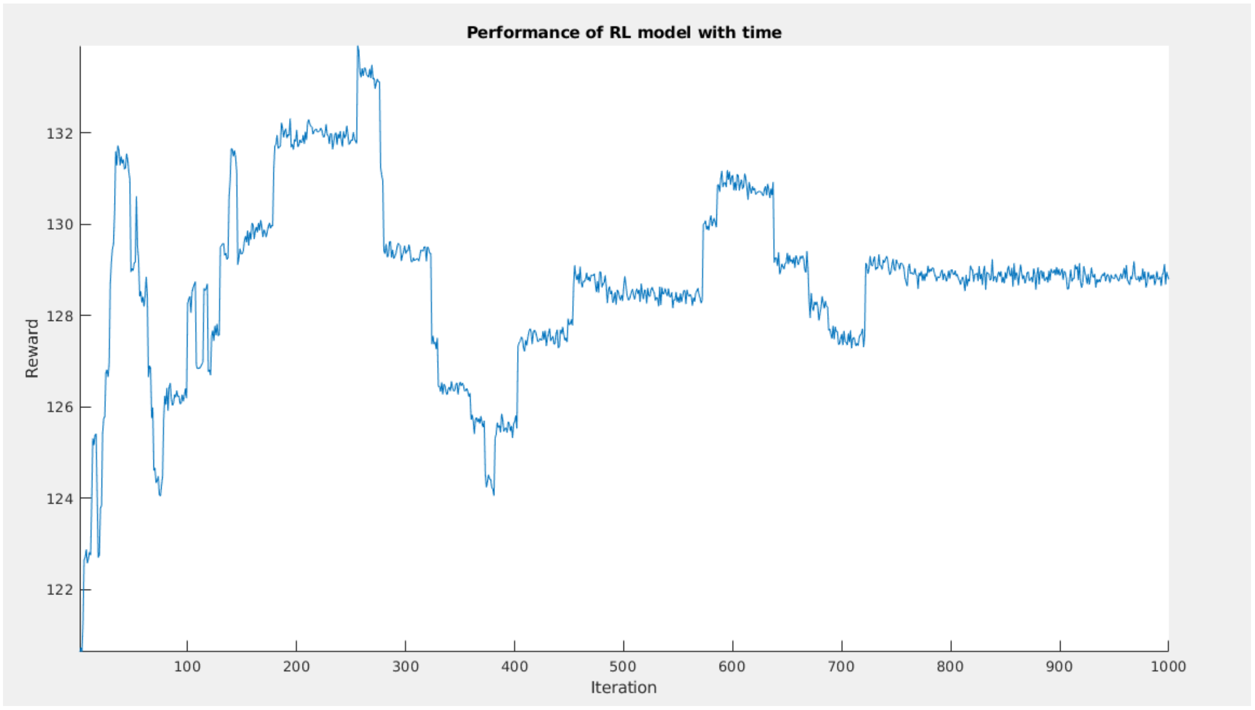
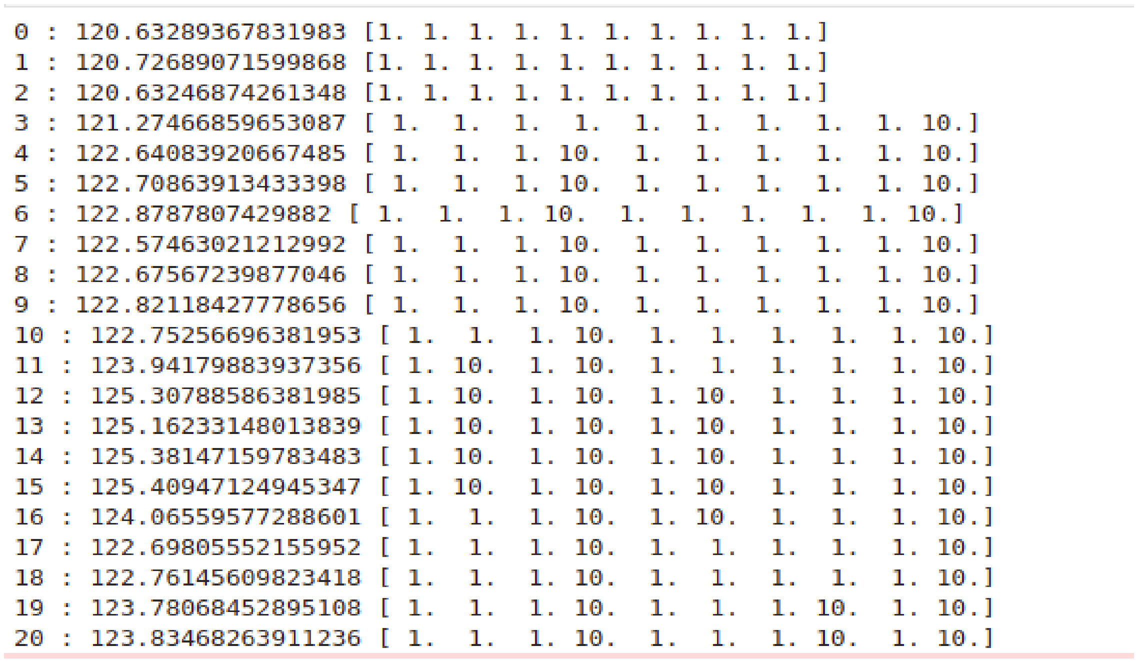
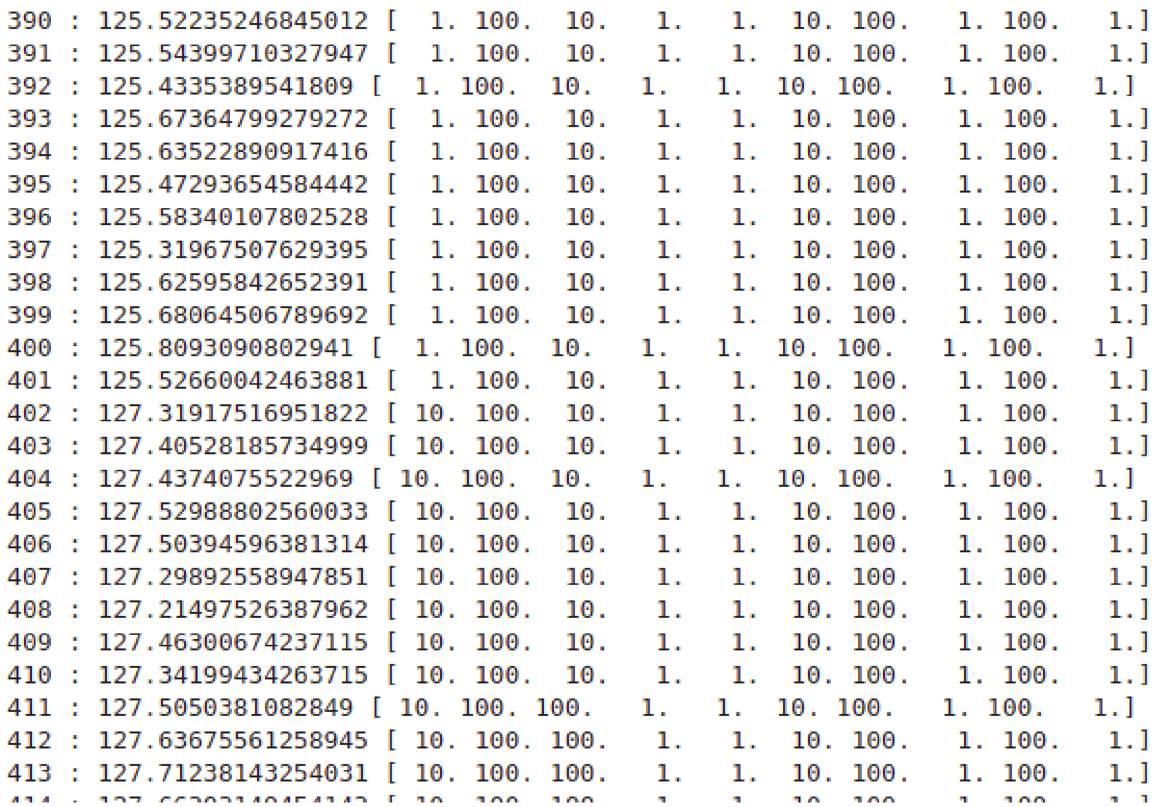


Fig1: Spectral Efficiency vs Number of Iterations



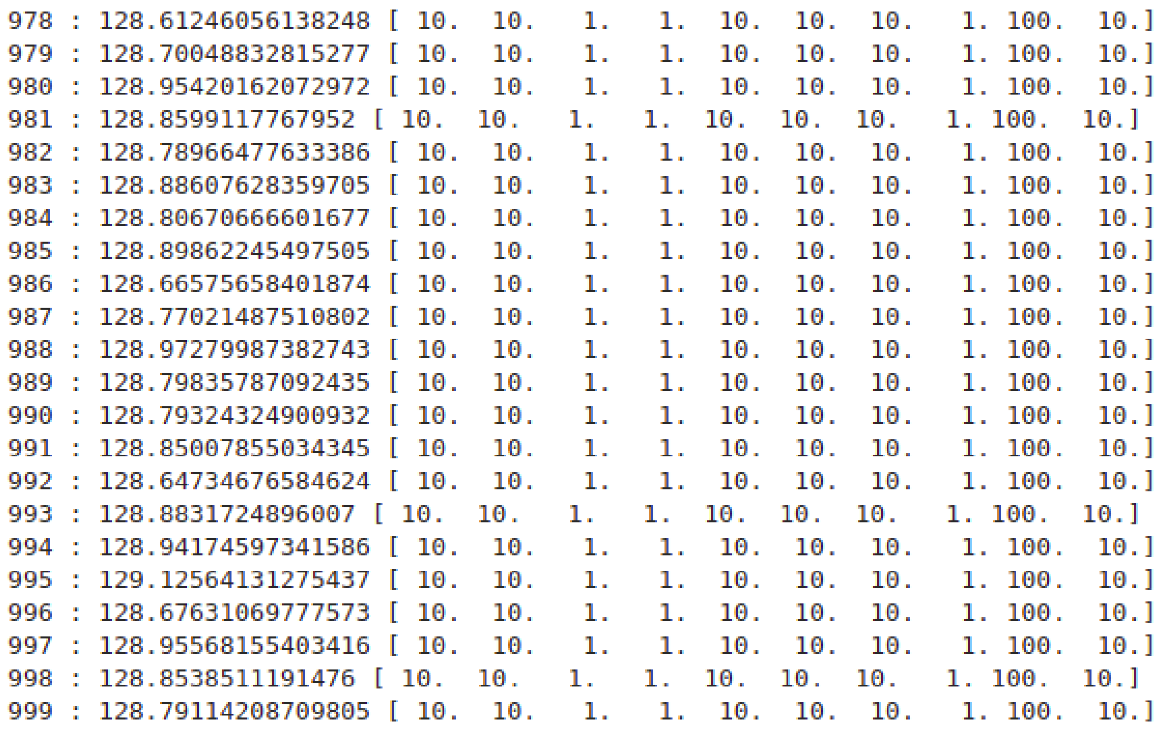
```
0 : 120.63289367831983 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
1 : 120.72689071599868 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
2 : 120.63246874261348 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
3 : 121.27466859653087 [ 1.   1.   1.   1.   1.   1.   1.   1.   1. 10.]
4 : 122.64083920667485 [ 1.   1.   1. 10.   1.   1.   1.   1.   1. 10.]
5 : 122.70863913433398 [ 1.   1.   1. 10.   1.   1.   1.   1.   1. 10.]
6 : 122.8787807429882 [ 1.   1.   1. 10.   1.   1.   1.   1.   1. 10.]
7 : 122.57463021212992 [ 1.   1.   1. 10.   1.   1.   1.   1.   1. 10.]
8 : 122.67567239877046 [ 1.   1.   1. 10.   1.   1.   1.   1.   1. 10.]
9 : 122.82118427778656 [ 1.   1.   1. 10.   1.   1.   1.   1.   1. 10.]
10 : 122.75256696381953 [ 1.   1.   1. 10.   1.   1.   1.   1.   1. 10.]
11 : 123.94179883937356 [ 1. 10.   1. 10.   1.   1.   1.   1.   1. 10.]
12 : 125.30788586381985 [ 1. 10.   1. 10.   1. 10.   1.   1.   1. 10.]
13 : 125.16233148013839 [ 1. 10.   1. 10.   1. 10.   1.   1.   1. 10.]
14 : 125.38147159783483 [ 1. 10.   1. 10.   1. 10.   1.   1.   1. 10.]
15 : 125.40947124945347 [ 1. 10.   1. 10.   1. 10.   1.   1.   1. 10.]
16 : 124.06559577288601 [ 1.   1.   1. 10.   1. 10.   1.   1.   1. 10.]
17 : 122.69805552155952 [ 1.   1.   1. 10.   1.   1.   1.   1.   1. 10.]
18 : 122.76145609823418 [ 1.   1.   1. 10.   1.   1.   1.   1.   1. 10.]
19 : 123.78068452895108 [ 1.   1.   1. 10.   1.   1.   1. 10.   1. 10.]
20 : 123.83468263911236 [ 1.   1.   1. 10.   1.   1.   1. 10.   1. 10.]
```

Fig2: Initial Values(Starting with low rewards)



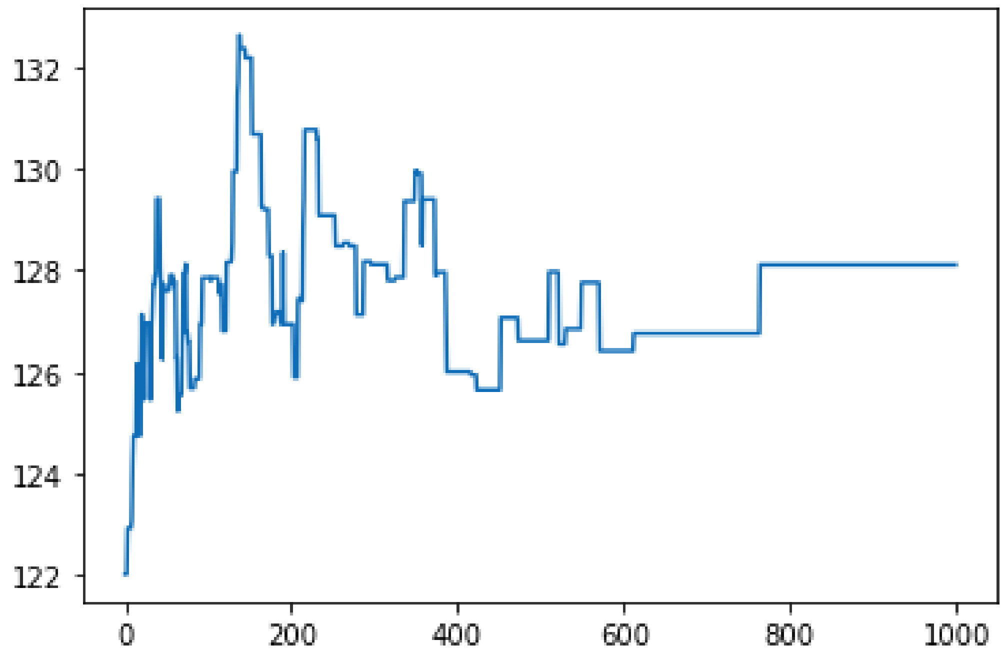
```
390 : 125.52235246845012 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
391 : 125.54399710327947 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
392 : 125.4335389541809 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
393 : 125.67364799279272 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
394 : 125.63522890917416 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
395 : 125.47293654584442 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
396 : 125.58340107802528 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
397 : 125.31967507629395 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
398 : 125.62595842652391 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
399 : 125.68064506789692 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
400 : 125.8093090802941 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
401 : 125.52660042463881 [  1. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
402 : 127.31917516951822 [ 10. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
403 : 127.40528185734999 [ 10. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
404 : 127.4374075522969 [ 10. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
405 : 127.52988802560033 [ 10. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
406 : 127.50394596381314 [ 10. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
407 : 127.29892558947851 [ 10. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
408 : 127.21497526387962 [ 10. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
409 : 127.46300674237115 [ 10. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
410 : 127.34199434263715 [ 10. 100.  10.   1.   1.  10. 100.   1. 100.   1.]
411 : 127.5050381082849 [ 10. 100. 100.   1.   1.  10. 100.   1. 100.   1.]
412 : 127.63675561258945 [ 10. 100. 100.   1.   1.  10. 100.   1. 100.   1.]
413 : 127.71238143254031 [ 10. 100. 100.   1.   1.  10. 100.   1. 100.   1.]
```

Fig3: Values in around the middle (Fluctuating states and values)



```
978 : 128.61246056138248 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
979 : 128.70048832815277 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
980 : 128.95420162072972 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
981 : 128.8599117767952 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
982 : 128.78966477633386 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
983 : 128.88607628359705 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
984 : 128.80670666601677 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
985 : 128.89862245497505 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
986 : 128.66575658401874 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
987 : 128.77021487510802 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
988 : 128.97279987382743 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
989 : 128.79835787092435 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
990 : 128.79324324900932 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
991 : 128.85007855034345 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
992 : 128.64734676584624 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
993 : 128.8831724896007 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
994 : 128.94174597341586 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
995 : 129.12564131275437 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
996 : 128.67631069777573 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
997 : 128.95568155403416 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
998 : 128.8538511191476 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
999 : 128.79114208709805 [ 10.  10.   1.   1.  10.  10.  10.   1. 100.  10.]
```

Fig4: Values towards the end converging to a particular value and state



Output in Python

Fig5: Plot for Spectral Efficiency vs Number of Iterations in a noiseless environment

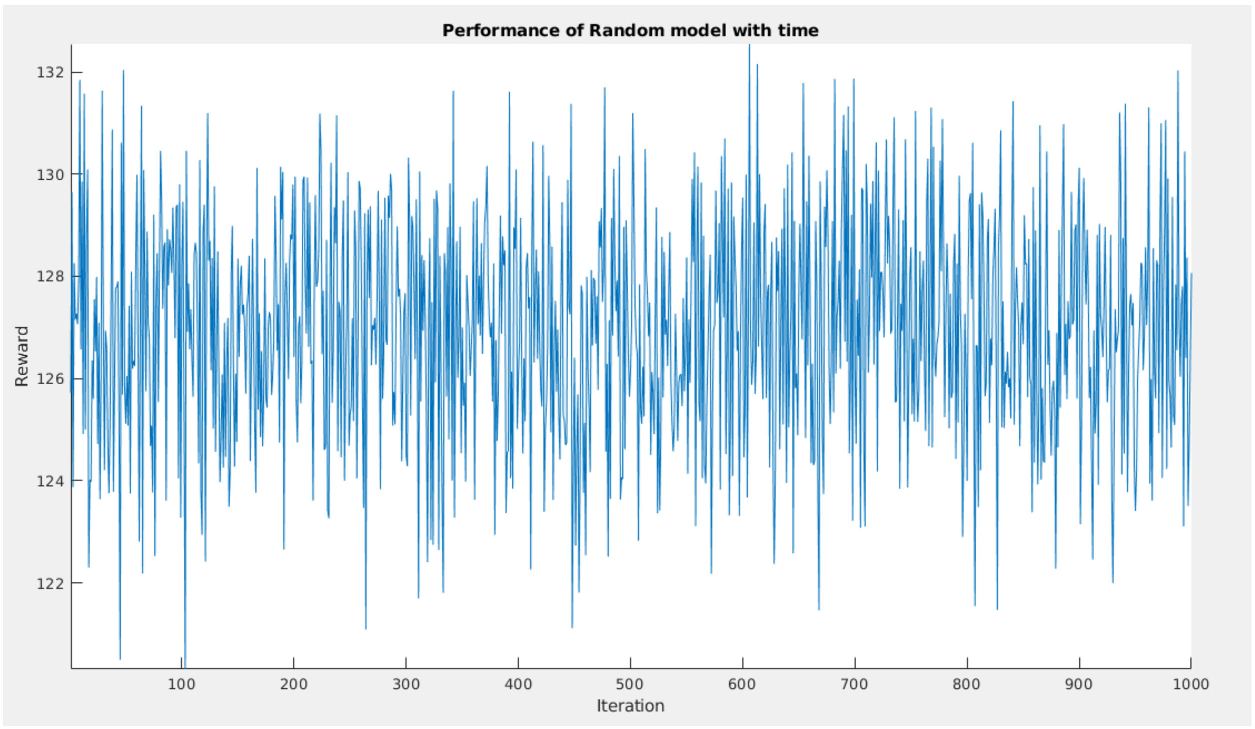


Fig6: Plot for Spectral Efficiency vs Number of Iterations for a random model

Observations:

In the Spectral efficiency Vs Number of iterations plot as shown in Figure1, we can observe that, initially we get less reward and gradually as the reinforcement model is about to learn, we observe a corresponding growth in the rewards Further, in the in the early stage, we see that the reward we get fluctuates a lot and towards the end, it seems to stabilize. This pattern can be attributed to the exploration-exploitation strategy which we have incorporated in our model. We have used epsilon-greedy methodology as our exploration-exploitation strategy which enables our model to explore states more often in the initial stage to get more varied experience and on further iterations it starts to rely more on it's past experience to get better rewards which can be also termed as exploiting is it's available knowledge. This behaviour helps the model to converge to an optimum state while also exploring a wide number of actions so that it doesn't get stuck on a local maxima in terms of rewards. The reward values corresponding to the states depicted in figures 2,3,4 show the same behaviour. Figure 5 illustrates the plot for Spectral Efficiency vs Number of Iterations in a noiseless environment which can be thought of as an ideal setup, thus, we see towards the end we get a straight line with no variations. On the contrary, if we observe Figure 6, we can observe that the reward can vary from 120 to 133 if we randomly assign power to the UEs. Thus, it shows that a model trained as shown in figure 1 constantly gives a reasonably good reward, in our case it sets to 128 which is quite close to 133 the optimum value we could find out by running 1000 iterations. Due to large state space, it is computationally tough to check for all power values, thus, we feel the random 1000 set of values' optimum value can be considered a good approximation for the absolute optimum value.

2)Algorithm devised

Reinforcement learning code:

```
#Imports
import numpy as np
import matplotlib.pyplot as plt
from collections import deque
import tensorflow as tf
from tensorflow import keras
import random
import matlab.engine

eng=matlab.engine.start_matlab()

#Global Variables
EPISODES = 1
TRAIN_END = 0


#Hyper Parameters
def discount_rate(): #Gamma
   return 0.95

def learning_rate(): #Alpha
   return 0.001

def batch_size(): #Size of the batch used in the experience replay
   return 20

R=[]

class DeepQNetwork():
   def __init__(self, states, actions, alpha, gamma, epsilon,epsilon_min, epsilon_decay):
      self.nS = states
      self.nA = actions
      self.memory = deque([], maxlen=250000)
      self.alpha = alpha
      self.gamma = gamma
      #Explore/Exploit
      self.epsilon = epsilon
      self.epsilon_min = epsilon_min
      self.epsilon_decay = epsilon_decay
```

```
        self.model = self.build_model()
        self.loss = []

    def build_model(self):
        model = keras.Sequential()
        model.add(keras.layers.Dense(24, input_dim=self.nS, activation='relu'))
        model.add(keras.layers.Dense(24, activation='relu'))
        model.add(keras.layers.Dense(self.nA, activation='linear'))

model.compile(loss='mean_squared_error',optimizer=keras.optimizers.Adam(lr=self.alpha))
        return model
    import math

    def sigmoid(x):
        return 1 / (1 + math.exp(-x))
    def action(self, state):
        if np.random.rand() <= self.epsilon:
            return np.random.randint(0,(int(nA/3))),np.random.randint(0,3)
        action_vals = self.model.predict(state)
        temp=np.array(action_vals[0]);
        action_2d=np.reshape(temp, (int(nA/3), 3));
        action_val_arg=np.argmax(action_2d, axis=1);
        action_val_value=np.amax(action_2d, axis=1);
        action_max_arg=np.argmax(action_val_value);
        return action_max_arg,action_val_value[action_max_arg]

    def test_action(self, state):
        temp=np.array(action_vals[0]);
        action_2d=np.reshape(temp, (int(nA/3), 3));
        action_val_arg=np.argmax(action_2d, axis=1);
        action_val_value=np.amax(action_2d, axis=1);
        action_max_arg=np.argmax(action_val_value);
        return action_val

    def store(self, state, action, reward, nstate, done):
        self.memory.append( (state, action, reward, nstate, done) )

    def experience_replay(self, batch_size):
        minibatch = random.sample( self.memory, batch_size )
        x = []
        y = []
        np_array = np.array(minibatch)
        st = np.zeros((0,self.nS))
        nst = np.zeros((0,self.nS))
        for i in range(len(np_array)):
            st = np.append( st, np_array[i,0], axis=0)
            nst = np.append( nst, np_array[i,3], axis=0)
        st_predict = self.model.predict(st)
        nst_predict = self.model.predict(nst)
        index = 0
        for state, action, reward, nstate, done in minibatch:
            x.append(state)
            nst_action_predict_model = nst_predict[index]
            nst_action_predict_model_2d = np.reshape(nst_action_predict_model, (int(nA/3), 3));

            target=np.zeros(int(nA/3));
            if done == True:
                target = target + reward
            else:
                target = reward + self.gamma * np.amax(nst_action_predict_model);
            qtobeupdated= np.argmax(nst_action_predict_model);
            target_f = st_predict[index]

            target_f[qtobeupdated]=target;

            y.append(target_f)
            index += 1

        x_reshape = np.array(x).reshape(batch_size,self.nS)
        y_reshape = np.array(y)
        epoch_count = 1
        hist = self.model.fit(x_reshape, y_reshape, epochs=epoch_count, verbose=0)
        for i in range(epoch_count):
            self.loss.append( hist.history['loss'][i] )
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay

#Create the agent
nS = 10
nA = 3*nS
dqn = DeepQNetwork(nS, nA, learning_rate(), discount_rate(), 1, 0.001, 0.995 )

batch_size = batch_size()

basepower=1;
powerratio=10;
numberoflevels=3;
maxpower=basepower* (powerratio ** (numberoflevels -1));
```

```
#Training
rewards = []
epsilons = []
Rew = []
TEST_Episodes = 0
for e in range(EPISODES):
    state = np.zeros(nS)
    state=state+basepower;
    nstate = np.zeros(nS)
    tot_rewards = 0
    for time in range(1000):
        action_val_loc,action_val = dqn.action(state)
        state = np.reshape(state,nS);
        nstate = np.reshape(nstate,nS);
        nstate=state;
        if action_val==2:
          nstate[action_val_loc]=state[action_val_loc]*powerratio;
        elif action_val==0:
          nstate[action_val_loc]=state[action_val_loc]/powerratio;
        else:
          nstate[action_val_loc]=state[action_val_loc];
        nstate[nstate<basepower]=basepower;
        nstate[nstate>maxpower]=maxpower;

        xnstate= matlab.double(nstate.tolist())
        reward= eng.fxn(xnstate);
        print(time,':',reward,nstate)
        R.append(reward)
        Rew.append(reward)
        done =False

        tot_rewards += reward
        state = np.reshape(state, [1, nS])
        nstate = np.reshape(nstate, [1, nS])
        dqn.store(state, action_val, reward, nstate, done)
        state = nstate
        if done or time == 1000:
            rewards.append(tot_rewards)
            epsilons.append(dqn.epsilon)
            print("episode: {}/{}, score: {}, e: {}"
                  .format(e, EPISODES, tot_rewards, dqn.epsilon))
            break
    if len(dqn.memory) > batch_size:
        dqn.experience_replay(batch_size)

import csv
with open('Rew','w') as ff:
    write = csv.writer(ff)
    write.writerow(Rew)
plt.plot(Rew)

eng.quit()
```

Algorithm of Reinforcement Learning (Pseudo code)

```
i)    Import Matlab Engine
ii)   Initialise Hyper Parameters like Gamma , Alpha ,Batch Size.
iii)  Initialize DQN Agent Parameters like No. of States, no of Q values, Learning
      Rate , Discount Rate ,etc.
iv)   Initialize the Environment Parameters like basepower , No of levels ,Power Ratio
      ,etc
v)     for episode = 1, M do
          (1) Initialise State Vector , Next State Vector and Total_Rewards.
          (2) for t = 1, T do
                  (a) Get the Action Vector (at) from the Agent.
                  (b) Apply the Action at in the Matlab Simulation Code and obtain next
                      state (φt+1) and the Reward Value (rt).
                  (c) Store transition (φt, at, rt, φt+1) in D and Perform Gradient
                      Descent after forming the expected Q matrix which is formed by
                      absorbing the reward.
                  (d) If (Sizeof(D)>Threshold)
                          (i)   Sample random minibatch of transitions (φj , aj , rj , φj+1)
                                from D and perform experience Replay.
                  (e) Append the Reward (rt) value to the Reward storing vector.
          (3) end for
          (4) Plot the Reward vs t graph.
vi)   end for
```

Reinforcement Learning algorithm explanation:

Reinforcement learning (RL) [9] technique is a reward based learning that depends on the interaction with the environment. Here, the agent learns its behaviour based on feedback from the environment and subsequently tries to improve its action. The basis of solving reinforcement learning problem is to find a policy mapping from state to action that maximizes the accumulated reward. In general, a reinforcement learning problem is modelled using a fine Markov decision process (FMDA) consisting of five entities, i.e., state, action, reward, policy, and value. Q Learning is a reinforcement learning algorithm which uses Q (stands for Quality) function to estimate reward values. For any FMDP, Q-learning identifies an optimal action selection policy with the objective of maximizing the expected value of the total reward. It uses Temporal Difference (TD) learning. Temporal Difference value can be understood as an estimate of the amount of reward which is expected in the future. If the TD value is very small, it means that the classifier has understood the environment well and there is little scope for further improvement. Hence, the major goal of Q-learning is to minimize the TD values.

The Equation for updating Q value is stated as follows:-

$Q\ new(st, at) \longleftarrow Q(st, at) + \alpha * (rt + \gamma * maxaQ(st+1, a) - Q(st, at))$

where, st is the state at time t, at is the action taken at time t, rt is the reward obtained at time t, α is the learning rate, γ is the discount factor, $Q(st, at)$ is the old Q value, $maxaQ(st+1, a)$ is the estimate of optimal future value,

$[rt + \gamma * maxaQ(st+1, a)]$ is the TD target, and

$[rt + \gamma * maxaQ(st+1, a) - Q(st, at)]$ is the TD equation.

A major limitation of Q-learning is that it works only in the environments that have discrete and finite state-action spaces. In order to extend Q-learning to complex environments, we use Deep Neural Networks as function approximators that can learn the value function by taking just the states as inputs. Here, we refer to a complex environment as systems where the size of the state-action table is very large and thereby, it is infeasible to store this table. The Deep Q Learning is one such solution for applying the concept of Q Learning in such complex environments. It uses Deep Neural Networks to estimate the Q values of all possible actions at a given state.

The loss function of Deep Q-Learning is given by

$\text{loss} = (r + \gamma * \text{maxa} (Q\hat{}(s, a0) - Q(s, a))2$

where, s is the state, a is the action taken, r is the reward obtained, γ is the discount

factor and Qˆ(s, a0) is the delayed Q function. We can notice that the loss function is very similar to the TD function in the case of Q Learning.

3)Environment

Overview of the 5G network environment concepts used

- Our work entailed designing scalable Cell-Free 5G Massive MIMO network.
- Cell-Free Massive MIMO is a special case of the DCC framework from the Network MIMO literature.
- The current DCC framework for Network MIMO can be termed truly scalable after filling in many missing details such as dealing with initial access, pilot assignment, and channel estimation/
- In the paper we based our simulation of the environment on is "E. Björnson and L. Sanguinetti, [1] where they mention the three step process for master AP allocation, pilot assignment and dynamic allocation of additionally serving APs.
- Further, they extensively discuss about how the complexity for calculations should be independent of the number of UEs because as the number of UEs tend to infinity, the computational complexity of such calculations (like Uplink combining vector and downlink precoding vector computation) will blow up leading to making these algorithms non-scalable. The computational complexity of three step process is ensured to be independent of K.
- The various Uplink combining vector and downlink precoding vector evaluation techniques along with new estimation algorithms keeping in mind the scalability of the system as proposed in paper has been used in our simulation. Their

complexities have been presented in the table below.

Scheme	Channel estimation	Combining vector computation
MMSE	$(N\tau_p + N^2)K\lvert\mathcal{M}_k\rvert$	$\frac{(N\lvert\mathcal{M}_k\rvert)^2+N\lvert\mathcal{M}_k\rvert}{2}K + (N\lvert\mathcal{M}_k\rvert)^2 + \frac{(N\lvert\mathcal{M}_k\rvert)^3-N\lvert\mathcal{M}_k\rvert}{3}$
P-MMSE	$(N\tau_p + N^2)\lvert\mathcal{P}_k\rvert\lvert\mathcal{M}_k\rvert$	$\frac{(N\lvert\mathcal{M}_k\rvert)^2+N\lvert\mathcal{M}_k\rvert}{2}\lvert\mathcal{P}_k\rvert + (N\lvert\mathcal{M}_k\rvert)^2 + \frac{(N\lvert\mathcal{M}_k\rvert)^3-N\lvert\mathcal{M}_k\rvert}{3}$
LP-MMSE	$(N\tau_p + N^2)\sum\limits_{l\in\mathcal{M}_k}\lvert\mathcal{D}_l\rvert$	$\frac{N^2+N}{2}\sum\limits_{l\in\mathcal{M}_k}\lvert\mathcal{D}_l\rvert + \left(\frac{N^3-N}{3} + N^2\right)\lvert\mathcal{M}_k\rvert$
MR	$(N\tau_p + N^2)\lvert\mathcal{M}_k\rvert$	$-$

- The combining vector formula for LP-MMSE is as follows:

$$\mathbf{v}_{kl}^{\mathrm{LP-MMSE}} = p_k \left(\sum_{i\in\mathcal{D}_l} p_i \left(\widehat{\mathbf{h}}_{il}\widehat{\mathbf{h}}_{il}^{\mathrm{H}} + \mathbf{C}_{il} \right) + \sigma^2 \mathbf{I}_N \right)^{-1} \widehat{\mathbf{h}}_{kl}.$$

- The Signal-to-Interference-plus-Noise rRatio (SINR) and Spectral Efficiency(SE) formulas used in our code respectively is as follows:

$$\mathsf{SINR}_k^{(\mathrm{ul},2)} = \frac{p_k \left|\mathbb{E}\left\{\mathbf{v}_k^{\mathrm{H}}\mathbf{D}_k\mathbf{h}_k\right\}\right|^2}{\sum\limits_{i=1}^{K} p_i \mathbb{E}\left\{\left|\mathbf{v}_k^{\mathrm{H}}\mathbf{D}_k\mathbf{h}_i\right|^2\right\} - p_k\left|\mathbb{E}\left\{\mathbf{v}_k^{\mathrm{H}}\mathbf{D}_k\mathbf{h}_k\right\}\right|^2 + \sigma_{\mathrm{ul}}^2\mathbb{E}\{\|\mathbf{D}_k\mathbf{v}_k\|^2\}}.$$

$$\mathsf{SE}_k^{(\mathrm{ul},2)} = \frac{\tau_u}{\tau_c}\log_2\left(1 + \mathsf{SINR}_k^{(\mathrm{ul},2)}\right)$$

- In the paper they had mentioned power allocation can influence the performance of the systems and thus in our work, we tried to develop an optimised allocation algorithm that can enhance the performance of the system by negating the amount of interference among channels and Pilot signals while keeping the spectral efficiency high. Also, our algorithm has been tailored to be scalable with respect to number of UEs.
- In the paper, they had given the comparison of performance of various combining and precoding scheme by providing the spectral efficiency graphs for a given set of environment parameters which showed that the scalable shemes LPMMSE outperform the MR (conjugate beamforming). Thus, we have used LPMMSE technique in our environment to test our power allocation algorithm.

.

Environment setup parameters

- Number of APs per setup is 50 and Number of antennas per AP is 2.
- Number of UEs we have used is 10 randomly spread across over an area of 20m x 20m.
- There is a single channel present between each ue-ap pair.
- In the code we maintain a set of diagonal matrices $D_{il} \in C_{N \times N}$, for i = 1,...,K and l = 1,...,L, determining which AP antennas may transmit to which UEs. More precisely, the jth diagonal element of D_{il} is 1 if the jth antenna of AP l is allowed to transmit to and decode signals from UE i and 0 otherwise.

3 step process as per the code

Gain over noise db is calculated for each channel. A ue k is assigned a master ap for which gain over noise db is the highest. The pilot assignment is done sequentially for the first tau p ues, then subsequently the pilot with least interference is assigned to the ue. Then the spatial correlation matrix is computed between the given ue and all aps. Then Each AP serves the UE with the strongest channel condition on each of the pilots where the AP isn't the master AP, but only if its channelEach AP serves the UE with the strongest channel condition on each of the pilots where the AP isn't the master AP, but only if its channel is not too weak compared to the master AP.

In summary, the UE appoints the AP with the strongest large-scale fading channel coefficient as its Master AP and it is assigned to the pilot that this AP observes the least pilot power on.

The concept of D mat is mentioned in the DCC concept and is borrowed from there.

Here

$$\begin{aligned}\widehat{s}_k &= \sum_{l=1}^{L} \mathbf{v}_{kl}^{\mathrm{H}} \mathbf{D}_{kl} \mathbf{y}_l^{\mathrm{ul}} \\ &= \mathbf{v}_k^{\mathrm{H}} \mathbf{D}_k \mathbf{h}_k s_k + \sum_{i=1, i \neq k}^{K} \mathbf{v}_k^{\mathrm{H}} \mathbf{D}_k \mathbf{h}_i s_i + \mathbf{v}_k^{\mathrm{H}} \mathbf{D}_k \mathbf{n}\end{aligned}$$

$s_i \in \mathbb{C}$ is the signal transmitted from UE i

$D_{il} \in \mathbb{C}^{N\times N}$, for i = 1,...,K and l = 1,...,L, determining which AP antennas may transmit to which UEs.

vkl is combining vector

hk is collective channel from all APs

$$y_k^{\mathrm{dl}} = \sum_{l=1}^{L} \mathbf{h}_{kl}^{\mathrm{H}} \sum_{i=1}^{K} \mathbf{D}_{il}\mathbf{w}_{il}\varsigma_i + n_k = \mathbf{h}_k^{\mathrm{H}} \sum_{i=1}^{K} \mathbf{D}_i\mathbf{w}_i\varsigma_i + n_k.$$

$\varsigma_i \in \mathbb{C}$ is the independent unit-power data signal intended for UE i

$w_{il} \in \mathbb{C}^N$ denote the precoder that AP l assigns to UE i.

4)Future work

Working on the second problem statement which is to compute D (UE-AP association matrix) vector using reinforcement learning.The objective however remains the same that is to shift the cumulative distribution function of spectral efficiency plotted with respect to UEs is towards the right as much as possible. Or in other words, maximise the spectral efficiency of the system. But here the optimization parameter is different, here it will be an association matrix instead of power matrix.
Constraint: Each ue should get at least one ap. D(i,k) =1 for at least one ith ap for a given kth ue.
Components of the reinforcement learning algorithm:
1) **Agent:** Central Server
2) **Action:** Change parameters of D. Each step corresponds to change in the association of ue and ap.
3) **Reward:** area above the curve for attaining the best performance.
4) **State:** D matrix

Once, we are able to solve both the problem statements individually, as our next step we plan to integrate them together to develop comprehensive algorithm able to handle both the objectives simultaneously.

For this we are planning to implement a similar model with similar architecture but different action space. The major problem that we are anticipating is sparse action space that might lead to very slow learning and increase the computation time as well. Furthermore , we are also aiming for a scalable solution by leveraging the concept of federated learning . Each UE will have a separate agent that will be responsible for power allocation level of the corresponding UE and it will also be responsible for the AP association matrix corresponding to the particular UP. In limited time intervals, the global agent will learn from the knowledge gathered by the local agents. This will ensure scalability as well as reduce the overall convergence period of the agents.

5)References

1. “E. Björnson and L. Sanguinetti, "Scalable Cell-Free Massive MIMO Systems," in IEEE Transactions on Communications, vol. 68, no. 7, pp. 4247-4261, July 2020, doi: 10.1109/TCOMM.2020.2987311.”
2. Van Chien, Trinh et al. “Large-Scale-Fading Decoding in Cellular Massive MIMO Systems With Spatially Correlated Channels.” IEEE Transactions on Communications 67 (2019): 2746-2762.
3. Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.
4. Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236
5. Structured Massive Access for Scalable Cell-Free Massive MIMO Systems,Shuaifei Chen and Jiayi Zhang and Emil Bjrnson and Jing Zhang and Bo Ai,IEEE Journal on Selected Areas in Communications,year-2021, volume-39, pages-1086-1100.
6. Playing Atari with Deep Reinforcement Learning Volodymyr Mnih et al @ deepmind.com
7. Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.
8. Bengio, Yoshua. Learning deep architectures for AI. Now Publishers Inc, 2009.
9. R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: A Bradford Book, 2018.