Internship Report

Nathan Boyer

July 26, 2024

Abstract

It's possible to distribute the Internet to users via drones. However it is then necessary to place the drones according to the positions of the users. Moreover, the 5th Generation (5G) New Radio (NR) technology is designed to accommodate a wide range of applications and industries. The NGNM 5G White Paper [5] groups these vertical use cases into three categories:

1. enhanced Mobile Broadband (eMBB)
2. massive Machine Type Communication (mMTC)
3. Ultra-Reliable Low-latency Communication (URLLC).

A partitioning of the physical network into several virtual networks looks to be the most suitable approach to provide a customized service to each application and to limit the operation expenses. This design is well known as *network slicing*. Each drone must thus slice its bandwith between each of the 3 user classes. This whole problem (placement + bandwidth) is often defined as an optimization problem, but since it is very hard to solve efficiently, it is almost always addressed by AI. In my internship, I wanted to prove that viewing the problem as an optimization problem can still be useful, by building an hybrid solution involving on one hand AI and on the other optimization. I use it to achieve better results than approaches that use only AI, although at the cost of slightly larger (but still reasonable) computation times.

1 Probleme presentation and literature analysis

The concept of network slicing has been investigated for some time. With the purpose of providing multiple core networks over the same network infrastructure, DECOR and eDECOR [1] have been already used in legacy LTE networks. However, the slicing of Radio Access Network (RAN) was not considered in these works yet. Since then, the 5G network slicing concept evolved, by providing a better modularization and flexibility of the network functions. RAN slicing is now of great interest for the literature, given the ability to support resource isolation among different slices, providing a way to allocate radio resources to the connected User Equipments (UEs) based on the slices they belong to. Several RAN slicing approaches have been proposed in the past few years. Traditional strategies ([3], [2]) mainly deal with orthogonal resource allocation in time and/or frequency domain, such that the isolation between different services is guaranteed. Their aim in mainly oriented at allocating virtual Resource Blocks (vRBs) to UEs for intra-slice scheduling and to map the allocated vRBs to Physical Resource Blocks (PRBs) at a second stage.

Machine Learning (ML) techniques ([6], [4]), mainly deep reinforcement learning, have also been extensively considered, given their capability to find patterns within the huge amount of data that is exchanged in cellular networks.

2 Problem solving by constrained optimization

2.1 In-depth explanation

This problem can be solved by constrained optimization ([4]).

First, we have to define the equation that computes how well a user will reveive a base station's connection depending on where it is.

We have the following Equations for the drones's SINRs, which is a factor that represents how much data is effectively receiveed by the user, in relation to the data sent by the base station to the user:

$$\mathrm{SINR}_{i,j}^{t} = \frac{pc\mu\left(y_j, u_i^t\right)\left(\left(\left\|y_j - u_i^t\right\|\right)^2 + (h)^2\right)^{-\alpha/2}}{\sum\limits_{k \in \mathcal{U} \setminus i} pc\mu\left(y_j, u_k^t\right)\left(\left(\left\|y_j - u_k^t\right\|\right)^2 + (h)^2\right)^{-\alpha/2} + \sigma^2}$$

To evaluate a configuration we will thus proceed as follows:

You first need to associate each user with its nearest base station.

You then share the bandwidth of a base station dedicated to a given class between all users of this class associated to this base station.

Then, you compute each user SINR and decide for the user is satisfied or not.

The percentage of satisfied user is thus what you want to maximize.

2.2 Optimization problem

This problem can be resolved by as an optimization problem, I will show you how to resolve it for 2 base stations.

Let's first define u a matrix of dimensions $N * N$, with N being the number of positions possible for a base station. $u_{i,j}$ is a binary variable that is equal to one if and only if the first base station is in position i and the second is position j.

bw is a $3 * 2$ matrix that represents the bandwidth each base station allocates to each user class.

δ is a vector that represents for every whether it is satisfied or not.

The function to optimize is then: $\max_{u,bw} \sum_{g\in\mathcal{G}} \delta_g$

Under the following constraints:

1. $\sum_{b\in|\mathcal{U}|} \sum_{i,j\in\mathcal{U}} u_{i,j} \times bw_{\text{slice}(g),b} \times \text{bps}_g \frac{100}{G_{\text{conn}}(\text{slice}(g),b)} \geq th_g \times \delta_g \qquad \forall g \in \mathcal{G}$

2. $bw_{em,b} + bw_{ur,b} + bw_{mm,b} = 1 \qquad \forall b \in |\mathcal{U}|$

3. $bw_{em,b} \geq 0; bw_{ur,b} \geq 0; bw_{mm,b} \geq 0 \qquad \forall b \in |\mathcal{U}|$

4. $\sum_{i,j\in\mathcal{U}} u_{i,j} = 1$

5. $\delta_g \in \{0,1\} \qquad \forall g \in \mathcal{G}$

6. $u_{i,j} \in \{0,1\} \qquad \forall i,j \in \{0,1,\ldots,|\mathcal{U}-1|\}$

The resolution of this optimization problem can be implemented in Python with the help of a module such as pulp.

2.3 Time Optimizations

To speed up the resolution of this problem, I implemented a few optimizations:

First, only positions in the convex enveloppe of the users can actually be candidates for optimal placements of the base stations. We can thus reduce by a lot the size of the matrix u.

Moreover, it is possible to go further but not without losing precision. Indeed, you can divide the users into k clusters and take the union of this convex enveloppe of this clusters.

You can visualize the transformation done as follows:

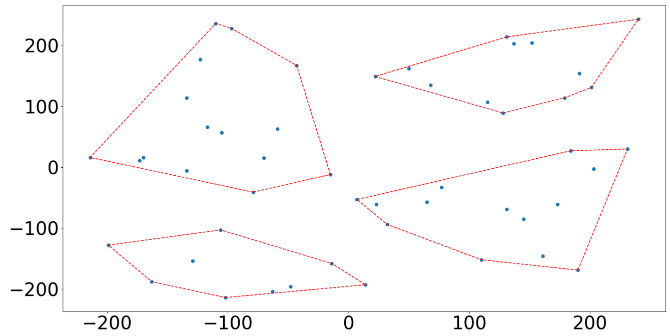

Figure 1: Representation of the convex enveloppe of the 4 clusters

The graph of performance and time in function of the number of clusters can be found below.

Moreover, I am using *Quote article here* to generate instances, thus as the instances generated have as many clusters as they have base stations, I am going to generate instances using clusters too. The performances are then as follows:

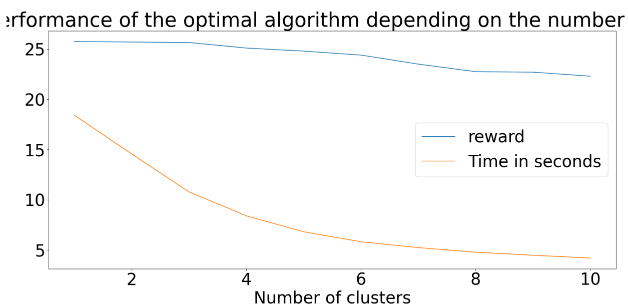

Figure 2: Performance when users are random

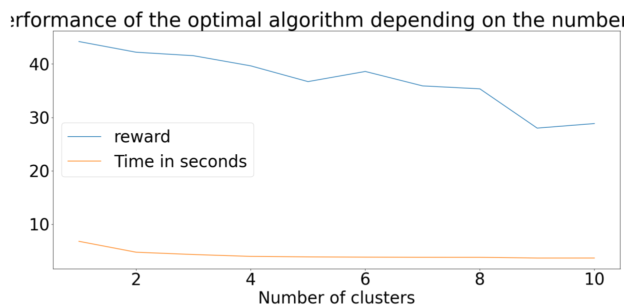

Figure 3: Performance when users are in 2 clusters

However, since the computation times are pretty reasonable for 2 base stations, I decided to go with only 1 cluster.

Finally, multithreading can be used to further accelerate the computations.

3 Machine learning to solve the problem

However, a major issue is that this method of solving the problem by optimization is still pretty long. This can be problematic, especially if the users are moving, and it thus becomes necessary to recalculate the optimal position often.

Thus, a solution using machine learning was considered. Indeed, solving an instance of the problem would only require a forward pass of the neural network, which is negligible in time.

3.1 Reinforcement learning

This is why a reinforcement learning solution was imagined and implemented by Lorenzo Bellone. *Quote article*

He tried to solve the same problem as the one I presented and, to put it shortly, managed to get an average reward of 72-73% of users satisfied.

However, there are multiple differences between how he approached this problem and how I did:

1. First, he moves the base stations setp by step and computes the mean reward over 100 movements. However, since the base stations spend most of their times idling (because they have already reached their best possible position), I took the bias of putting the base stations directly at the computed best position, and then computing the reward only once (which also allows to test the different agents over more instances in the same amount of time).

2. Secondly, what is plotted in his article is the average reward during the reinforcement process. What it means is that the neural network is only tested on the instances it just learnt from. What I did instead is I have a separate test set, which I use to evaluate the different agents.

3.2 Supervised learning from the optimal solution

But, an other idea is to use supervised learning. By using an optimal agent, the neural network can learn to replicate the optimal solutions.

The neural network is then divided in 2 neural networks:

1. A neural network that takes as inputs the positions of the users and learns to output the best positions for the 2 base stations.

2. A neural network that takes as inputs the positions of the users and the positions of the base stations and learns to output the best slicing of bandwiths between the 3 classes of users.

As a side note, the type of each users is represented by 3 one-hots, one for each of the class.

To conclude with my results, here is the average error for the neural network that learns to output the positions: As well as the average error for the neural network that learns to output the bandwiths:

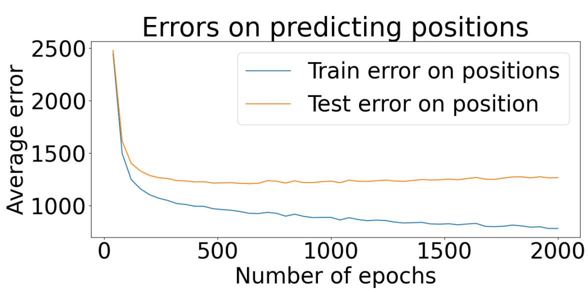

Figure 4: Position error

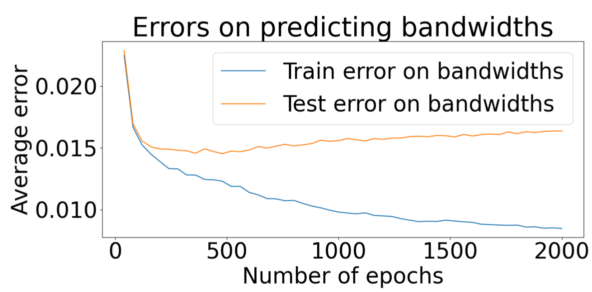

Figure 5: Bandwidth error

With these neural networks, we can test it on instances and evaluate the performances it gives.

(We plot here the normalized average user coverage, i.e. the average user coverage divided by the average user coverage of the optimal solution)

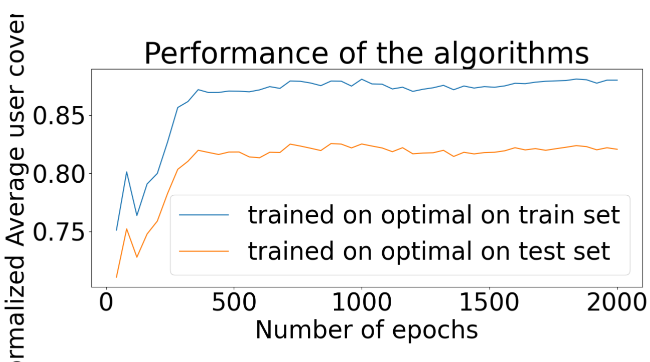

Figure 6: Representation of the convex enveloppe of the 4 clusters

3.3 Learn only what's necessary

However, you can see the test error on bandwiths stops to decrease rapidly. That's why I decided to plot the performances when the positions are decided optimally, but the bandwiths are decided with the AI.

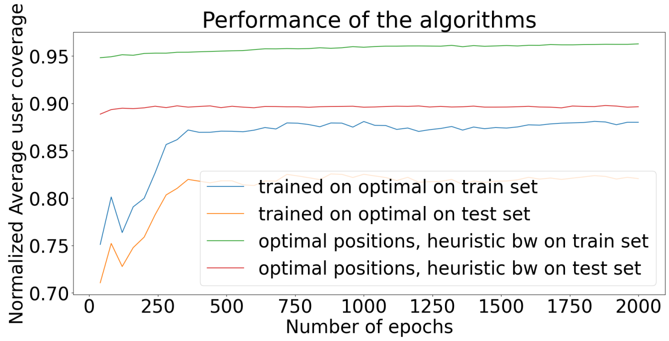

Figure 7: Performance with optimal positions and AI bandwiths

As we can see, the bandwidth part seems to be the issue.

Thus, I implemented an agent which decides positions with a neural network, but decides the bandwiths optimally, which is pretty fast once the positions are fixed.

We then have the following results:

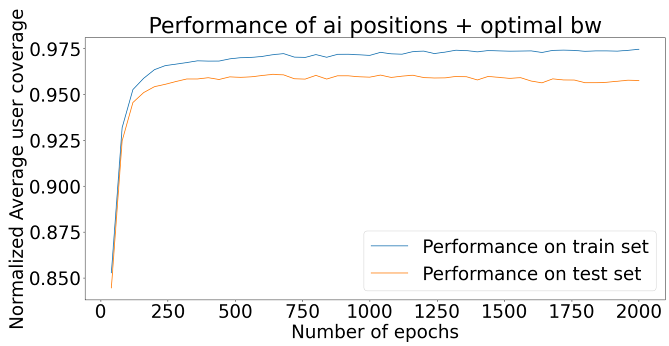

Figure 8: Performance with AI positions and optimal bandwiths

To do a complete comparaison of the different methods, I computed the mean rewards and time to execute of the 3 methods quoted until now:

Type of agent	Average user coverage	Execution time
AI positions and bandwiths	76.80%	0.7875 ms
AI positions and optimal bandwiths	84.72%	15.677 ms
Optimal positions and bandwiths	87.08%	2.2759 s

3.4 Generalizing to a variable number of users

However, until now, I have only resolved instances with a fix number of users.

To fix this, the first solution we imagined was to use graph neural networks. *Detail how gnn work*

The graph given to the neural network was constructed using a k-neighbourg algorithm. I have made a graphic plotting the final test error in function of the k in the k-neighbourg algorithm.

However, as you can see, whatever the chosen k is, the results are always lacking.

I thus opted for a simpler but less beautiful approach: I just add a one-hot to each user stating if the drone exists or not. Thus, the neural network can be used to resolve instances with a variable number of users, as long as the number in question is inferior to a certain maximum.

The precision is very similar to what it used to be with a fix number of users as you can see in the following graphics:

Put some graphics here

4 Ouverture: Utiliser la distance à l'optimum comme erreur pour l'apprentissage

5 Conclusion

References

[1] TR 23.711 3GPP. Enhancements of dedicated core networks selection mechanism (release 14), September 2016.

[2] Xenofon Foukas, Mahesh Marina, and Kimon Kontovasilis. Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture. In *unknown*, pages 127–140, 10 2017.

[3] Adlen Ksentini and Navid Nikaein. Toward enforcing network slicing on ran: Flexibility and resources abstraction. *IEEE Communications Magazine*, 55:102–108, 06 2017.

[4] Emiliano Traversi Lorenzo Bellone, Boris Galkin and Enrico Natalizio. Deep reinforcement learning for combined coverage and resource allocation in uav-aided ran-slicing. *IEEE*, 7:104279–104293, 2019.

[5] NGMN. NGMN 5G white paper. *NGMN Alliance*, page 125, 2015.

[6] Yi Zhou, Zhanqi Jin, Huaguang Shi, Zhangyun Wang, Ning Lu, and Fuqiang Liu. Uav-assisted fair communication for mobile networks: A multi-agent deep reinforcement learning approach. *remote sensing*, 2022.