A learning-based scheme for channel allocation to vehicular users in wireless networks

Thi Thuy Nga Nguyen, Olivier Brun, Balakrishna Prabhu

A Learning-based Scheme for Channel Allocation to Vehicular Users in Wireless Networks[⋆,⋆⋆]

Thi Thuy Nga Nguyen[a,1,*], Olivier Brun[b], Balakrishna J. Prabhu[b]

[a] *Torus Actions SAS, Toulouse, France*
[b] *LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France*

Abstract

Resource allocation algorithms in wireless networks can require solving complex optimization problems at every decision epoch. For large scale networks, when decisions need to be taken on time scales of milliseconds, using standard convex optimization solvers for computing the optimum can be a time-consuming affair that may impair real-time decision making. In this paper, we propose to use Data-driven and Deep Feedforward Neural Networks (DFNN) for learning the relation between the inputs and the outputs of two such resource allocation algorithms that were proposed in [1, 2]. On numerical examples with realistic mobility patterns, we show that the learning algorithm yields an approximate yet satisfactory solution with much less computation time.

Keywords: Scheduling, Deep Feedforward Neural Networks, Supervised Learning, Data-Driven.

1. Introduction

In cellular wireless networks, a central scheduling problem is to choose one among several concurrent users (for example, mobile phones) to which the scheduler (henceforth also referred to as the base station) must send data to. This scheduling decision is taken every time-slot which is of the order of 2 ms [4] and is based on what are called as channel conditions of the users. Roughly, the channel condition of a user determines the data rate at which the base station can communicate with this user. In wireless networks, these conditions can vary randomly on short as well as on long time scales. Also called fading and shadowing, these random variations are a consequence of the interference patterns

⋆This work was funded in part by a contract with Continental Digital Services France.
⋆⋆A preliminary version of this paper appeared in [3].
*Corresponding author
Email addresses: nttnga@torus-actions.fr (Thi Thuy Nga Nguyen), brun@laas.fr (Olivier Brun), bjprabhu@laas.fr (Balakrishna J. Prabhu)
[1]The work of this author was done when she was employed by Continental Digital Services France.

induced by the different obstacles (building, trees, etc.) in the path of the radio waves used for wireless communications [5]. However, the decision is not as easy as scheduling the user with the best channel condition as such a policy could lead to unfairness between users. Imagine a user with a direct line of sight path with the base station and another who is inside a building or an underground metro station. Quite possibly, the former user will always have a better channel which would starve the latter user of any communication.

To avoid these unfair allocations, a typical solution is to define a utility which is usually a concave function of the throughput[2] of the users and then compute the scheduling decision as the one that maximizes the sum of the utilities of the users. For example, the utility function could be the logarithm of the average throughput of the users. These solutions fall under the umbrella of the network utility maximization problem [6].

In an ideal scenario, the base station will solve this utility maximization problem every 2 ms. However, there are two practical issues that make this infeasible. First, the throughput of a user depends upon future channel conditions which are unknown to the scheduler. Second, the integer scheduling problem is known to be NP-complete [4]. Solving such an optimization problem over a time horizon of seconds (thousands of time-slots) and with hundreds of users can be unrealistic in time-slots of 2 ms (which are actually becoming shorter as the technology progresses). To overcome these practical issues, several heuristics have been proposed that are based on an estimation of the future data rates [4, 1, 2]. The heuristics (called STO1 and STO2) in [1, 2] use the estimated future data rates as an input to a relaxed version of the original problem restricted to a shorter time horizon thereby reducing the dimensionality of the problem. These heuristics are thus much faster to solve than the original problem but they still require solving rather frequently a large-scale convex optimization problem which can be time consuming. It was shown numerically in [1, 2] that STO1 and STO2 performed better than the one in [4] as well as the popular PF algorithm [7] that does not use future estimated rates. Therefore, in this paper, we shall address the problem of improving the speed of these heuristics without compromising on their superior total utility. It is worth mentioning that although we only focus on STO1, our approach could probably be extended to other channel allocation algorithms as well.

1.1. Contributions

We propose a machine learning based solution to speed up the operations of STO1[3]. The key idea is to use a Deep Feedforward Neural Network (DFNN) to approximate the output of the relaxed optimization problem in the heuristics.

[2]We use data rate and throughput to denote two different but related quantities. The throughput only takes into account the data rate of the time-slots in which a user is served. It is thus no more than the total data rate of this user.

[3]STO2 is similar to STO1 but solves the optimization problem less frequently. In this paper we shall focus on STO1 but the ideas developed here can be applied to STO2 as well.

It is well-known that any continuous function can be approximated arbitrary well by a DFNN. However, discontinuous functions are much more difficult to learn. Unfortunately, as we show with a simple example, the input-output mapping realized by the STO1 algorithm is discontinuous. This makes the model slower to learn due to oscillations at discontinuity points. We characterize the discontinuity points of STO1 by explicitly indicating the set where they reside when the dimension is equal to 2. In order to reduce the impact of discontinuities, we propose several user ordering schemes. We also propose an alternative approach which amounts to learning the dual values, which are proven mathematically to be continuous.

It will be shown on numerical examples that once the DFNN is trained, it takes much less time to generate a reasonable accurate solution compared to using the specialized Python package CVXPY [8] that uses the solver Mosek [9] for solving of a convex optimization problem. We prune and compare different DFNN architectures and different loss functions to find the most appropriate ones for our problem. We then compare the behavior of the learning algorithm with STO1 and with other existing algorithms as well. The comparison shall be done on scenarios created with SUMO [10] which can generate realistic mobility patterns of vehicles on road networks. Based on numerical results, the learning algorithm is shown to perform close to STO1 with much less computation time. Numerical results in section 5.6 show that, for larger scaling systems, only learning dual values could scale well in comparison with learning the primal values (either with or without ordering scheme).

Preliminary results from this paper mainly containing the numerical examples appeared in [3].

1.2. Related works

The theory of approximation for DFNNs has been studied in many papers. Motivated by Kolmogorov's superposition theorem [11] in 1957, many approximation results have proven the approximation capabilities of feedforward neural networks for the class of continuous functions, see, e.g., [12],[13],[14]. In his theorem, Kolmogorov proved that any continuous function can be represented as a superposition of continuous functions of one variable. In 1989, Cybenko proved that any multivariate continuous function with support in a hypercube can be uniformly approximated by a linear finite combinations of compositions of a sigmoidal functions and a set of affine functions [12]. This representation is in fact a feedforward neural networks with sigmoidal activation functions. Independently with the work of Cybenko, Hornik [13] also proved a similar result. Two years later, Hornik [15] showed that multi-layer feedforward neural networks with arbitrary bounded and non-constant activation function can approximate arbitrary well real-valued continuous functions on compact subsets of $\mathbf{R}^n$ as long as sufficiently many hidden layers are available. The adjective "deep" in "deep learning" thus simply means many layers.

Learning an algorithm to produce an approximate algorithm in order to reduce computation time has been proposed in several recent research papers [16], [17]. In [16], the authors consider a Sparse Coding problem which is used

for extracting features from raw data. The problem is that Sparse Coding is often too slow for real-time processing in several applications such as pattern recognition. The authors propose a method using a non linear, feedforward function to learn Sparse Coding to produce an approximate algorithm with 10 times less computation.

Learning an algorithm for wireless resource management has been proposed in [17]. In that work, the authors used a DFNN to learn an algorithm for the interference channel power control problem. They obtain an almost real time algorithm, since passing the input through a DFNN to get the output only requires a small number of simple operations as compared to an iterative optimization algorithm. They show that, by choosing an appropriate initialization, the initial power control algorithm performs a continuous mapping which can be efficiently learnt.

In this paper, we use DFNNs for learning a channel allocation algorithm maximizing the proportional fairness between vehicular users. The proposed method is however potentially applicable to other convex optimization problems.

1.3. Organization

In Section 2, we present the resource allocation problems considered in this paper in the case of a single Base Station (BS). We also briefly describe the STO1 algorithm for allocating the channel to vehicular users. Section 3 provides an overview of the learning-based approach and formally defines the input-output relationship for the DFNN model. We discuss about the continuity of this input-output mapping in Section 4 and then propose several user ordering schemes for reducing the discontinuity of the original mapping, as well as an alternative approach which amounts to learning the dual values instead of the allocation. Numerical results are presented in Section 5. Finally, in Section 6 we discuss several research directions that can be followed in future work.

2. Wireless Scheduling Problems and Channel Allocation Heuristics

We first present the wireless scheduling problems considered in this paper in Section 2.1, and then briefly describe the channel allocation heuristics that were proposed in [1, 2] in Section 2.2.

2.1. Problem Formulation

Consider the following downlink discrete-time channel allocation problem for a single Base Station (BS) (see [7] and references therein):

$$\begin{cases} \max\limits_{\alpha} \quad O(\alpha) = \sum\limits_{i=1}^{K} \log\left(\sum\limits_{j=1}^{T} \alpha_{ij} r_{ij}\right) \\ \text{subject to} \\ \qquad \sum\limits_{i=1}^{K} \alpha_{ij} \leq 1, \quad j = 1, \ldots, T; \\ \qquad \alpha_{ij} \in \{0,1\}, \quad j = 1, \ldots, T, \ i = 1, \ldots, K. \end{cases} \tag{I}$$

Here $r_{ij} \geq 0$ is the data rate of user i in time-slot j, and α_{ij} is the corresponding allocation in this slot. The constraints impose that the BS can choose at most one user in each time-slot. The objective of the BS is to maximize the sum of the individual user utilities which are defined as the logarithm of the total throughput of the user over a time horizon of T. As mentioned in Section 1, solving this problem is not practical because the data rates $r_{i,j}$ become known to the scheduler only in slot j. Further, users arrive and leave and it is not possible to know in advance which users will be present in the network in the future. Hence, the algorithms proposed use either no information on the future rate (see [7] and references therein) or an estimation of the future rates [4, 1]. The latter works assume that the mean future rates of users can be predicted from their positions using Signal-to-Noise Ratio (SNR) maps.

A more fine-grained scheduling problem on the downlink involves joint power control and channel allocation which allows the BS to vary the transmit power in addition to choosing how much of the channel (or bandwidth) it can allocate to the users [18], and is defined as:

$$\begin{cases} \max\limits_{x,p} \quad \sum\limits_{i=1}^{K} \log\left(\sum\limits_{j=1}^{T} x_{ij} \log\left(1 + \frac{p_{ij}\gamma_{ij}}{x_{ij}}\right)\right) \\ \text{subject to} \ \sum\limits_{i=1}^{K} x_{ij} \leq 1, \ \forall j; \quad x_{ij} \geq 0, \forall i, j \\ \qquad \frac{1}{T} \sum\limits_{j} \sum\limits_{i} p_{ij} \leq \bar{P}; \quad \sum\limits_{i} p_{ij} \leq P_{max}, \ \forall j. \end{cases} \tag{II}$$

Here p_{ij} (resp. x_{ij}) is the power (resp. fraction of bandwidth) allocated by the BS to user i in slot j. The parameter γ_{ij} represents the channel condition of the user i in slot j and $x_{ij} \log\left(1 + \frac{p_{ij}\gamma_{ij}}{x_{ij}}\right)$ is the Shannon rate obtained by this user when it is allocated power p_{ij} and fraction of bandwidth x_{ij}. The utility of the user is again the logarithm of its total throughput and the objective of the BS is to maximize the sum utility. Note that there are constraints on the power allocation which involve both the maximum power that can be expended in a time slot as well as the average power spend over the whole horizon. For this joint power and channel allocation problem, an adapted version of STO1 was proposed in [2].

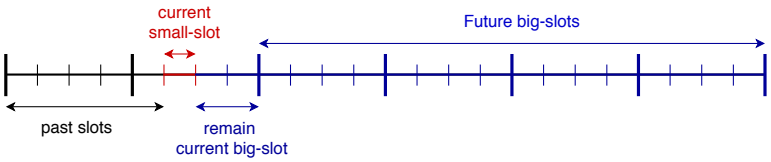


Figure 1: The different types of time slots in the STO1 algorithm.

Remark 1 (Joint power control and channel allocation). *For brevity, we shall explain the STO1 algorithm and its associated machine learning solution only for the channel allocation problem* (I)*. A remark shall be made wherever the treatment of this problem differs from that of the joint power control and channel allocation problem* (II)*.*

2.2. The STO1 algorithm

STO1 is a sequential algorithm which computes the allocation on time-slot j based on the current and the past data rates, and the estimated future data rates. It operates on two time-scales (shown in Figure 1) in order to reduce its complexity. Define big-slots as a certain number (order of hundreds) of time-slots (or small-slots) that reflect the duration over which the distance of each user to the BS does not change much. It follows from the law of large numbers that, although the data rate of each user in each time-slot can vary randomly, the expected sum of its data rate over big-slots does not vary too much. For example, while the scheduling time-slots are 2 ms in length, one big slot can be equal to 100 time-slots, and one user can move at most by a few meters in a big slot. Big slots are defined to reduce the dimension of the original optimization problem as explained below. It shall be assumed that estimations are available for users' future positions at the granularity of big-slots, and that the mean of future rates based on the future positions are estimated using SNR maps.

Before giving a formal definition of the STO1 algorithm, we give an intuitive explanation. In each small-slot j, STO1 does two steps. In the first step, it solves the problem (I) but with several restrictions: *(i)* the horizon is shortened to J big-slots; *(ii)* the future allocations are computed only on the aggregated level of big-slots; and *(iii)* the integer constraints on α_{ij} are relaxed. In the second step, the fractional allocation for the current small-slot is projected onto the set of the feasible integral allocations. The first step reduces the number of variables and hence the dimensionality of the problem as the future allocations are computed only for big-slots. Note that the second step is optional and is relevant only to problems with integral constraints.

A more formal definition is as follows. Denote by δ the length of a time slot. Let Δ be the size of the big-slot in absolute time units and let $m = \Delta/\delta$ be the number of small slots in a big-slot (see Figure 1). Denote by $\bar{r}_{ij}$ the mean rate in slot j for user i. At each small-slot t, with a slight abuse of notation, we shall denote by $\bar{\rho}_{i,0} = \sum_{j=t+1}^{(m-(t \mod m))+t} \bar{r}_{ij}$ the total rate for user i in the remaining channel allocation slots of the current big-slot $\tau = 0$, where $t \mod m$ denotes

the remainder when dividing t by m. We also define $\bar{\alpha}_{i,0}$ as the corresponding allocation for the current big-slot $\tau = 0$.

Denote by $\bar{\rho}_{i\tau} = \sum_{j=(\tau-1)m+A+1}^{\tau m+A} \bar{r}_{ij}$ where $A = \left(\lfloor \frac{t}{m} \rfloor + 1\right) m$, is the total average data rate that user i will get in the future big-slot τ ($\tau = 1, 2, ..., J-1$), where big slot τ starts after the current big-slot, and J is the short time horizon in term of big slots over which we can estimate the mean future rate. We also define $\bar{\alpha}_{i\tau}$ as the corresponding allocation for user i in future big slot τ. These allocations $\bar{\alpha}_{i\tau}$ can be interpreted as the fraction of small slots that user i will be allocated in the big-slot τ.

Note that this definition is slightly different from the definition in [1]. The differences are as follows: in [1], there is no current big slot and the future big-slot starts just after the current small slots. This change in definition does not change much the overall performance of the algorithms. The above definition of two time slot types corresponds in fact to the ones introduced in [2].

Denote by $a_i(t) = \sum_{j=1}^{t} \alpha_{ij} r_{ij}$ the total throughput allocated to user i up to time slot t, and let $K(t)$ be the number of users inside the coverage range of the BS at time t.

The algorithm STO1 contains two steps which are as follows:

- **Step 1**– solve the following optimization problem over a short-term horizon of J big-slots:

$$\begin{cases} \text{maximize} & \displaystyle\sum_{i=1}^{K(t)} \log\left(a_i(t-1) + \alpha_{it} r_{it} + \sum_{\tau=1}^{J} \bar{\alpha}_{i\tau} \bar{\rho}_{i\tau}\right) \\ \text{subject to} & \\ & \sum_{i=1}^{K(t)} \alpha_{it} = 1, \\ & \sum_{i=1}^{K(t)} \bar{\alpha}_{i\tau} = 1, \quad \tau = 0, \ldots, J-1, \\ & \alpha_{it}, \bar{\alpha}_{i\tau} \in [0,1], \quad \tau = 0, \ldots, J-1,\ i = 1, \ldots, K(t). \end{cases} \tag{STO1-Opt}$$

  The decision variables in Problem (STO1-Opt) are the channel allocations in the current small slot, α_{it}, and the channel allocations in the current and future big-slots, $\bar{\alpha}_{i\tau}$. Since the future allocations are only computed on the time-scale of big-slots, there is a reduction by factor m in the number of variables in (STO1-Opt).

- **Step 2** – obtain a feasible integral allocation from the fractional one. For example, set $\alpha_{i^* j} = 1$ with $i^* = \text{argmax}_i\, \alpha_{ij}$ and set it to 0 for the other users.

Remark 2. (STO1-Opt) *is solved thanks to the python package CVXPY [8] and the solver Mosek [9]. In [1], this optimization problem was solved using a projected gradient algorithm, since it allows to iteratively solve a convex optimization problem when the feasible set is a simplex or a Cartesian product of simplices, no matter how complex the objective function is as long as it is smooth and convex. The advantage of CVXPY [8] is that it can be used to generalize this*

idea to other convex optimization problems, with more complex constraints such as power control constraints or others QoS constraints (e.g. delay constraints).

Remark 3. *STO1 leans on the information about the number of users and users' channels to make decisions. In this paper, we assumed that the average channel gains in big slots with less error could be provided as input of STO1, this assumption is quite reasonable because of the law of large numbers; the number of users in the near future is not assumed to be known for the BS because that information might be more difficult to get. However, if we can get that information as input for STO1, then the variables for the new users just need to be added into the optimization problem (their channel gains are assigned to 0 before their arrival instant). This information can be updated in every small slot (every 2 ms) when STO1 computes a new allocation.*

3. Learning Algorithm

As presented in [1] and [2], STO1 performs better than existing algorithms. However it has to solve an optimization problem with a large number of variables and constraints frequently, and may not be able to run in real time for some complex scenarios. The main idea proposed in this paper is to train a DFNN to predict an approximate solution to (STO1-Opt) instead of using a specialized convex optimization package. We provide an overview of the approach in Section 3.1. We then formulate the learning problem in Section 3.2, where we define the input state, the target and the architecture of the neural network.

3.1. A Learning-based Approach for Wireless Scheduling

Our goal is to approximate the input-output mapping realized by STO1 with a DFNN (see [19] for background material on supervised learning with DFNN). Once the approximation function (that is, the DFNN) obtained, the output can be computed by feeding the DFNN with the input value, instead of solving an optimization problem. This simpler method is expected to work faster than the original algorithm. Obviously, the same idea could be used for other problems in order to obtain an approximate method which performs almost as well as the original algorithm but requires much less computing time.

More precisely, the STO1 algorithm can be seen as a function F that maps an input $x_n \in \mathcal{X}$ (a problem instance) to an output $y_n \in \mathcal{Y}$ (a channel allocation), where $\mathcal{X}$ and $\mathcal{Y}$ denote the input and output spaces of STO1, respectively. Given a bunch of examples of input-output pairs $(x_n, y_n)_{n=1}^N$ (the training data), our objective is to obtain another function $\hat{F} : \mathcal{X} \to \mathcal{Y}$ which is in the form of a DFNN (an example of DFNN is illustrated in Figure 2) and minimizes the empirical risk

$$R_{\text{erm}}(\hat{F}) = \frac{1}{N} \sum_{n=1}^{N} l(y_n, \hat{F}(x_n)),$$

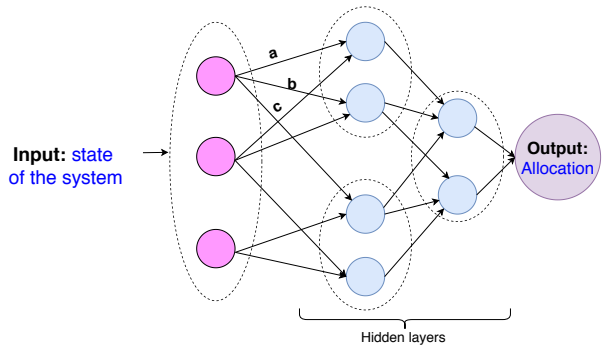


Figure 2: An example of Deep Feedforward Neural Network.

where $l(\cdot)$ is a loss function which measures how far y_n is from $\hat{F}(x_n)$. Our hope is that the learned approximation $\hat{F}$ is able to determine outputs of unseen inputs with small error.

It is known that optimizing the parameters (e.g., weight matrices, bias vectors, non-linear activation functions) of a neural network and deciding its architecture (e.g. numbers of layers and number of nodes in each layer) is in general not an easy task [20]. In this paper, we shall empirically compare some architectures through experiments presented in Section 5.2. After fixing the architecture, we have to find appropriate parameters by minimizing the empirical risk defined above.

3.2. System Setup for learning

Recall that, in STO1, we have two types of time scales: one is the big slot Δ and the other one is the small slot δ. In STO1, we solve problem (STO1-Opt) with variables of size $(1+J)\cdot K$ every small slot, where J is the time horizon in terms of big slots and K is the number of users in the system. The size of the allocation vector for each user is equal to $1+J$ since it contains the allocation for the current small slot, α_{it}, and the average allocation $\bar{\alpha}_{i\tau}$ for the subsequent J big slots (including the current big slot). The input of STO1 is a data rate vector of size $(1+J)\cdot K$ and the total allocated throughput for the K users. The output of STO1 is the current allocation vector $(\alpha_{it})_{i=1,\dots,K}$ which is of size K, since we shall only use current allocation for making decision.

As it is defined at the moment, STO1 is not well suited to be modelled as a learning problem for the two reasons stated below.

Firstly, since K can vary over time, the dimension of the input vector will also vary. To circumvent this problem and to properly define STO1 as a function, we have to fix the size of the state. To do that, we extend the real state of the system by adding some pseudo users. Let us assume that there are at most K_M users inside the system. We will then add $K_M - K$ pseudo users, where K is the number of real users in the system at time t. We will actually learn an

extended version of STO1 which is STO1 when we restrict it to K users. There are many ways to extend STO1, but here we try to define an extended version that preserves as much as possible the continuity of STO1. When we mention "learning STO1", it means "learning the extended function" of STO1.

Secondly, the output of STO1 as defined above is the solution of an optimization problem. So, STO1 can be a set-valued mapping since the solution need not be unique. However, by using the CVXPY package to solve the convex optimization problem (STO1-Opt), we let it decide the way it determines one of the solutions. This way STO1 becomes a function (instead of set-valued mapping).

Remark 4 (Joint power control and channel allocation). *For the joint power control and channel allocation problem, the state needs to be augmented by the remaining total power. The output of the DFNN will now give the transmit power to each user as well as the fraction of the channel it gets allocated.*

3.2.1. State

We define a state as a matrix of size $(2+J)\times K_M$, where K_M is the maximum number of users in the system. There are thus $2 + J$ rows, and each row has K_M elements. The interpretation is as follows:

- The first row gives the current rates of the K users. We fill in the K positions on the left hand side with these current rates, and the remaining $K_M - K$ positions are filled with -1.
- The next J rows (from $2, .., J + 1$) give the average rates of the users in the next J big slots. For pseudo users (the $K_M - K$ columns on the right hand side), we use the value $(-1) \cdot (\Delta/\delta - (t \mod \Delta/\delta))$ for the current big slot and the value $(-1) \cdot \Delta/\delta$ for the other big slots.
- The last row gives the total allocated throughput of the K users. For pseudo users, we use a large enough value which is significantly greater than the total allocated throughput of real users.

By observing how STO1 works, we remark, as expected, that STO1 gives priority to users with a low allocated throughput and a high current rate. Therefore, the way we define the state (that is, by using negative values for the current and future rates of pseudo users, and extremely large values for their allocated throughput) is intended to help the model ignore quickly the pseudo users.

Remark 5. *Remark that there are K real users in the system at present time. Therefore, the K places of the real users in the first row which give the current rates of those users have to be strictly positive. The future rates (from the second row to the $(J + 1)$-th row) can be zero.*

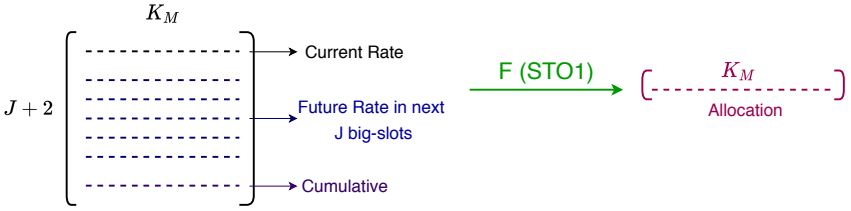


Figure 3: Input and output of the DFNN model.

3.2.2. Target

We remind the reader that we want to learn only the current allocation, not the future allocation. Therefore, the target will be a vector of size K_M, where the first K positions represent the fractional allocation α_{it} of the K users as computed by STO1, and the last positions are filled with zero. Since in the optimization problem, the sum of allocation should be equal to 1, when there is no user in the system (all positions correspond to pseudo users), the allocation vector will be set to $(1/K_M, ..., 1/K_M)$ by convention.

Figure 3 illustrates the input and output of the DFNN model as described above.

3.2.3. Loss, DFNN architecture, initial parameters and optimizer

We will try several different loss functions and architectures and compare them in Section 5. The initial parameters (weights) of the DFNN will be chosen as proposed in [21], which allows the initial parameters to be not too big and not too small. The optimizer is Adam, which was first introduced in [22] and is a stochastic first-order gradient-based algorithm. The convergence of Adam is proven in [23].

4. Discussion about the continuity of the STO1 function

As discussed in Section 1.2, any continuous function can be approximated arbitrary well with a DFNN. Unfortunately, as we shall prove in Section 4.1 below, the input-output mapping F implemented by the STO1 algorithm is not continuous, implying that it is much more difficult to learn than a continuous mapping. We characterize the discontinuities of this mapping in Section 4.2. This characterization is then used in Section 4.3 to devise user ordering schemes allowing to reduce the number of discontinuities. In Section 4.4, we propose an alternative approach which amounts to learning the dual values, which are proven to be continuous, instead of the primal ones.

4.1. A simple counter-example proving that STO1 is a discontinuous function

As defined above, F is a mapping from $\prod_{i=1,j=1}^{i=K_M,j=J+2} E_{i,j}$ to $\mathcal{S}_{K_M}$, where $\mathcal{S}_{K_M}$ is the simplex of size K_M and

- For $j = 1$ and $i = 1, \ldots, K_M$, $E_{i,1} = (0, +\infty] \cup \{-1\}$,

Figure 4: Two connected components of $E_{i,1} \forall i$.

- For $j = 2, \ldots, J+1$ and $i = 1, \ldots, K_M$, $E_{i,j} = [0, +\infty] \cup \{-1, -2, ..., -\Delta/\delta\}$,
- For $j = J+2$ and $i = 1, \ldots, K_M$, $E_{i,J+2} = [0, +\infty]$.

We can see that the pseudo users and real users lie in different connected components (see Figure 4), from which it follows that the continuity of its extension depends only on STO1. We shall thus concentrate on the continuity of the STO1 function on the connected component corresponding to the K real users. Unfortunately, STO1 itself is not a continuous function. Let us take a simple example illustrating the discontinuity. Assume that $K = 2$, $J = 1$, $\Delta = \delta$ and consider the following sequence of states:

$$R_n = \begin{pmatrix} r + \frac{(-1)^n}{n} & r - \frac{(-1)^n}{n} \\ r - \frac{(-1)^n}{n} & r + \frac{(-1)^n}{n} \\ 0 & 0 \end{pmatrix}.$$

Then it is easy to show that $R_n \to \bar{R}$, where

$$\bar{R} = \begin{pmatrix} r & r \\ r & r \\ 0 & 0 \end{pmatrix}.$$

However

$$F(R_n) = \begin{cases} [1, 0] \text{ if } n = 2k, \\ [0, 1] \text{ if } n = 2k + 1. \end{cases}$$

This simple example shows that the STO1 function is not continuous. The discontinuities make the learning problem much more difficult since near discontinuity points, the DFNN model does not know which direction of the output its parameters should follow. For example, as illustrated in the above 2-dimension example, near $\bar{R}$, the target (output) oscillates between $[0, 1]$ and $[1, 0]$. Since a DFNN is a composition of many continuous functions, it is also a continuous function. Trying to fit every target near $\bar{R}$ into the DFNN model can make its parameters conflict and result in a slower convergence. Therefore in the remainder this section we shall characterize the discontinuity points of the STO1 function and discuss how to derive a continuous function from STO1.

4.2. Analysis of discontinuities

Above we showed one discontinuity point $\bar{R}$ when $K = 2, J = 1$. Now, our objective is to characterize the set containing all discontinuity points. Before doing that, let us analyze the property of an optimal solution resulting from the

KKT conditions (which are necessary and sufficient conditions in our case). To simplify the notation, let us assume that a big slot is equal to a small slot, that is, $\Delta = \delta$ (the proof is however still valid if a big slot is equal to multiple small slots). With $\Delta = \delta$, we can write (STO1-Opt) as follows:

$$\left\{\begin{array}{l} \text{maximize} \sum_{i=1}^{K(t)} \log\left(\sum_{j=t}^{t+J} \alpha_{ij} r_{ij} + c_i\right) \\ \sum_{i=1}^{K(t)} \alpha_{ij} = 1, \quad j = t, t+1, ..., t+J, \\ \alpha_{ij} \in [0,1], \quad j = t, t+1, ..., t+J, \ i = 1, \ldots, K(t). \end{array}\right. \tag{I_R}$$

Let us denote by r_{ij} the rate of user i at time j, and let c_i be the total throughput allocated to user i up to present time. Consider the state R defined by

$$R = \begin{pmatrix} r_{1t} & r_{2t} & \cdots & r_{K_M t} \\ \cdots & \cdots & \cdots & \cdots \\ r_{1,t+J} & r_{2,t+J} & \cdots & r_{K_M,t+J} \\ c_1 & c_2 & \cdots & c_{K_M} \end{pmatrix},$$

in which the i^{th} column provides all the information available for user i. We denote the above optimization problem by (I_R) to emphasize that it depends on R.

The Lagrange function of this convex optimization problem is $L = O(\alpha) + \sum_j \lambda_j (1 - \sum_i \alpha_{ij}) + \sum_{i,j} \rho_{i,j} \alpha_{ij}$ where $O(\alpha) = \sum_{i=1}^{K(t)} \log\left(\sum_{j=t}^{t+J} \alpha_{ij} r_{ij} + c_i\right)$. The KKT conditions for problem (I_R) are as follows:

$$\left\{\begin{array}{l} \text{1, Primal feasibility: } \sum_i \alpha^*_{ij} = 1 \quad \forall j, \text{ and } \alpha^*_{ij} \geq 0 \quad \forall i, j, \\ \text{2, Dual feasibility: } \rho^*_{ij} \geq 0 \quad \forall i, j, \\ \text{3, Complementary slackness: } \alpha^*_{ij} \rho^*_{ij} = 0 \quad \forall i, j, \\ \text{4, Lagrange stationary: } \frac{r_{i,j}}{c_i + \sum_k \alpha^*_{ik} r_{ik}} = \lambda^*_j - \rho^*_{ij} \quad \forall i, j. \end{array}\right. \tag{KKT}$$

The last condition is implied by $\nabla_\alpha L(\alpha^*) = 0$.

From (KKT), it follows that if $\rho^*_{ij} > 0$, then $\alpha^*_{ij} = 0$ and $\frac{r_{i,j}}{c_i + \sum_k \alpha^*_{ik} r_{ik}} = \lambda^*_j - \rho^*_{ij} < \lambda^*_j$. It implies that α^*_{ij} is positive only when $\rho^*_{ij} = 0$ and in this case the ratio $\frac{r_{i,j}}{c_i + \sum_k \alpha^*_{ik} r_{ik}} = \lambda^*_j$ represents the maximum ratio one user i can get in slot j. Since $\sum_i \alpha^*_{ij} = 1$, there exists at least one user who gets a strictly positive allocation in time slot j, i.e, there exists at least one user i for which $\rho^*_{ij} = 0$. The output that we consider is only the current allocation in time slot t, therefore we shall discuss only the continuity of the allocation at time t. Let denote by $I_t(R)$ the set of users who have a strictly positive allocation in current slot, i.e, $I_t(R) = \{i \mid \rho^*_{it}(R) = 0\}$. By defining $I_t(R)$, we implicitly claim that ρ^*_{it} is defined uniquely by R and we shall prove later this uniqueness property. Consider the following sets

$$\mathcal{A} = \{R | \# I_t(R) = 1\} \text{ and } \mathcal{B} = \{R | \# I_t(R) \geq 2\}.$$

Then $\mathcal{A}$ and $\mathcal{B}$ represent a partition of the whole input space since $\#I_t(R) \geq 1$ for all states R as explained above. The set $\mathcal{A}$ is the set of problem instances for which there is exactly only one user having a strictly positive allocation at time t, that user thus get full allocation and others get nothing. The set $\mathcal{B}$ is the complement. Proposition 1 below states two fundamental properties of the set $\mathcal{A}$.

Proposition 1. *It holds that:*

- *the allocation $\alpha^*_{\mathbf{t}} = (\alpha^*_{it})_{i=1}^K$ is uniquely defined by R on $\mathcal{A}$,*
- *F: $R \mapsto \alpha^*_{\mathbf{t}} = (\alpha^*_{it})_{i=1}^K$ is continuous on $\mathcal{A}$, i.e, all potential discontinuities lie in $\mathcal{B}$.*

Proof. See Appendix A. ■

Remark 6. *Note that we do not claim that all points in the set $\mathcal{B}$ are points of discontinuity. We just claim that any discontinuity point necessarily belongs to the set $\mathcal{B}$.*

Proposition 1 shows that for all problem instances $R \in \mathcal{A}$, there exists a unique allocation α_t, and moreover that this allocation is continuous in R. However, the allocation is not necessarily unique for problem instances $R \in \mathcal{B}$, and it may even be discontinuous in this set, as shown by our simple example in Section 4.1.

4.3. Ordering schemes reducing the number of discontinuities

The counter-example in Section 4.1 exhibits a discontinuity point of F. We note however that if, by convention, we assume that users are numbered in decreasing order of current rates, then this discontinuity point disappears. This suggests that an appropriate ordering scheme could help reducing the number of discontinuities of F. As stated in Proposition 1, F is continuous on $\mathcal{A}$, so the set of all discontinuities is a subset of $\mathcal{B}$. It is difficult to provide further information on this set for the general case. Therefore, we shall first consider in this section a small 2-dimensional example for which the analysis is easier, in order to understand better the behaviour of the output of F and how an ordering scheme could make it continuous.

We restrict ourselves to the case $K = 2$, $J = 1$ and $t = 1$ and we further assume that the total allocated throughput of each one of the two users is equal to 0. It follows that the input R is of the following form:

$$R = \begin{pmatrix} r_{11} & r_{21} \\ r_{12} & r_{22} \\ 0 & 0 \end{pmatrix}$$

Figure 5 shows the structure of the sets $\mathcal{A}$ and $\mathcal{B}$, which is proven in Proposition 2.

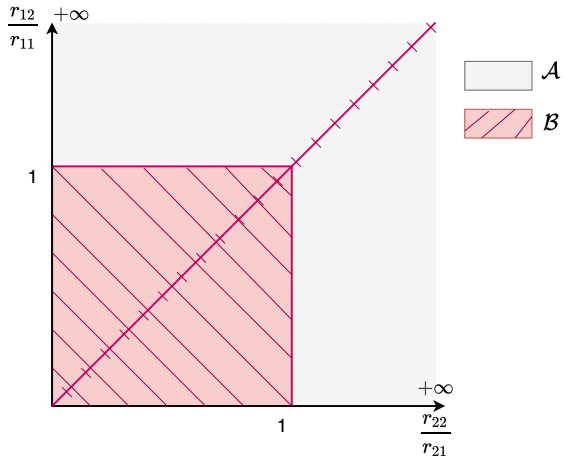


Figure 5: 2-dimension illustration for the sets $\mathcal{A}$ and $\mathcal{B}$.

Proposition 2. *With the axes shown in figure 5, we have the following claims:*

- $\mathcal{B}$ *contains the lines* $\frac{r_{22}}{r_{21}} = \frac{r_{12}}{r_{11}}$ *and the box defined by* $0 \leq \frac{r_{22}}{r_{21}} \leq 1$ *and* $0 \leq \frac{r_{12}}{r_{11}} \leq 1$.
- *On* $\mathcal{A}$*, the optimal current allocation is unique, continuous and integer.*
- *On the open box defined by* $0 \leq \frac{r_{22}}{r_{21}} < 1$ *and* $0 \leq \frac{r_{12}}{r_{11}} < 1$*, the optimal current allocation is unique, continuous but not integer, except for the inputs* R *on the line* $\frac{r_{22}}{r_{21}} = \frac{r_{12}}{r_{11}}$*. On the segment defined by* $\frac{r_{22}}{r_{21}} = 1$ *and* $\frac{r_{12}}{r_{11}} < 1$*, and on the segment defined by* $\frac{r_{12}}{r_{11}} = 1$ *and* $\frac{r_{22}}{r_{21}} < 1$*, the optimal current allocation is unique, continuous and integer.*
- *On the line* $\frac{r_{22}}{r_{21}} = \frac{r_{12}}{r_{11}}$*, the optimal current allocation is not unique and not continuous, no matter how we choose the current allocation among the set of optimal allocations.*

Proof. See Appendix B. ■

Figure 6 illustrates Proposition 2 by showing the different regions and the optimal solution in each region.

In this 2-dimensional case, the set of discontinuities (the line) is a small set in the sense that it has Lebesgue measure equal to 0. However it makes the model difficult to learn when the matrix input is near the line, since it oscillates between $\alpha^*_{\mathbf{1}} = (0, 1)$ and $\alpha^*_{\mathbf{1}} = (1, 0)$ outside the box and oscillates between $\alpha^*_{\mathbf{1}} = (\frac{1}{2} - \frac{1}{2}x, \frac{1}{2} + \frac{1}{2}x)$ and $\alpha^*_{\mathbf{1}} = (\frac{1}{2} + \frac{1}{2}y, \frac{1}{2} - \frac{1}{2}y)$ ($x \approx y$ and $x, y > 0$) inside the box. In order to obtain a continuous function, we propose to choose the optimal allocation when it is not unique as illustrated in Figure 7. Define $H(a_1, a_2) = (\max(a_1, a_2), \min(a_1, a_2))$. If we choose the solution as described in Figure 7, $H \circ F$ is a continuous function. It is reasonable to hope that learning $H \circ F$ is easier than learning the non-continuous function F directly.

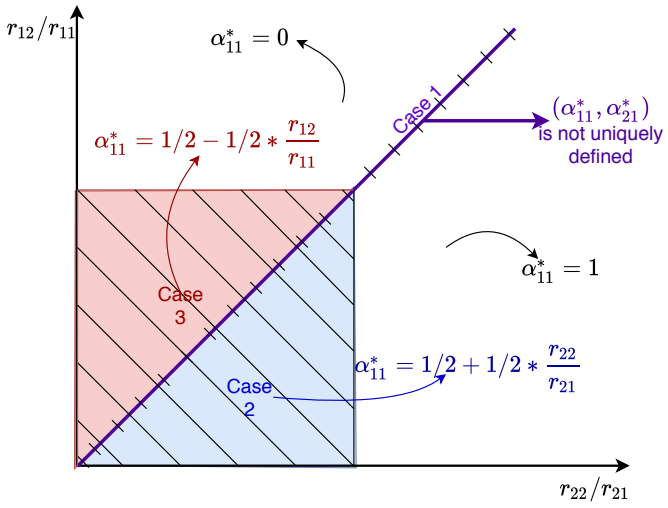


Figure 6: The output of STO1 in 2-dimension.

In general, H is an order function, i.e., $H(a_1, a_2, ..., a_K) = (a_{i_1}, a_{i_2}, ..., a_{i_K})$ such that $a_{i_1} \geq a_{i_2} \geq \cdots \geq a_{i_K}$. Once we learn $H \circ F$, we can recover F by storing the original places of all ordered elements. In more detail, we store the original places of all ordered elements by remembering the following function q:

$$k = q(i_k), \text{ for all } k = 1, 2, ..., K$$

In general, choosing an order in order to have a continuous function is not an easy task, since to choose an order as above we actually have to solve an optimization problem. We instead propose several heuristic orders, that are either based on current rate, based on the ratio between current rate and cumulative (the PF index), or based on the $(PS)^2S$ index [4]. The numerical results of these heuristic orders are shown in Section 5.4.

4.4. Learning the dual values

The user ordering schemes introduced in Section 4.3 provide a partial solution for reducing the negative impact of discontinuities on the learning time. In this section, we propose an alternative approach which is based on Proposition 3 below.

Proposition 3. *The optimal solution of the dual problem is such that:*

1. $\lambda^* = (\lambda_t, \lambda_{t+1}, ..., \lambda_{t+J})$ *is uniquely defined by* R *and continuous in* R.
2. $\rho^* = (\rho^*_{ij})_{i=1,j=t}^{i=K,j=t+J}$ *is uniquely defined by* R *and continuous in* R.

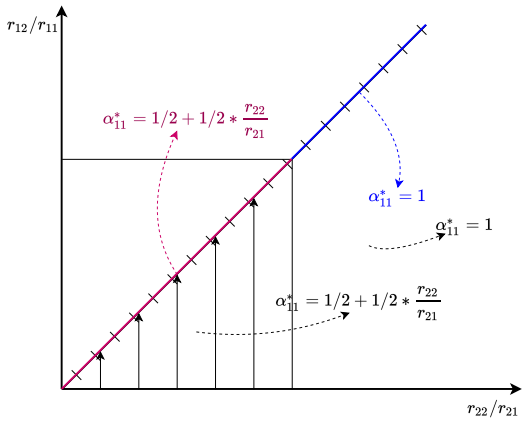


Figure 7: Choosing solution in the line to obtain a continuous function in 2-dimension.

PROOF. See Appendix C. ■

In summary, we know that the solution $\alpha^*_{\mathbf{t}}$ of (I_R) is not continuous in the input R, but that the dual variables λ^* and ρ^* are. We thus propose the following idea. Instead of learning the allocation vector $\alpha^*_{\mathbf{t}}$, we could learn the dual variable ρ^*. As mentioned above, the users who get a positive fraction of allocation $\alpha_{i,t} > 0$ all belong to the set $I_t(R)$. Therefore, after learning the dual values (ρ^*_{ij}), we propose to choose the current allocation as follows: $\alpha^*_{i^*(t),t} = 1$ and $\alpha^*_{i,t} = 0$ for all $i \neq i^*(t)$, where $i^*(t)$ is an index in $I_t(R)$. If there are more than one index in $I_t(R)$ we choose arbitrarily an index in that set. With the notations introduced in Appendix A, this is equivalent to first learning $\mathbf{C}^* = (C^*_1, C^*_2, ..., C^*_{K_M})$ and then choosing arbitrarily an index in the set argmax $\frac{r_{it}}{C^*_i}$.

5. Numerical Results

In this section, we do simulations to evaluate the influence of many factors on the behaviour of the DFNN model (loss functions, architecture of the DFNN, ordering schemes and learning dual values). We use the keras library [24] to implement our code.

There are actually a lot of factors that can have an impact on the behavior of the learning procedure such as the initial learning rate, the learning rate decay, the optimizer, the initial weight, the number of parameters, the activation functions in layers, an so on. Here we are not able to justify all our choices, but we focus on the factors which have the most significant impact on the learning algorithm in our opinion. The initial learning rate is chosen equal to 0.0015 and after each epoch, this learning rate decays by a factor 0.998.

5.1. A Unified Data Generator for Comparison

To support the comparisons in this section, the data (both for training and validation) is generated as follows. The number of users is generated randomly from 0 to $K_M = 10$. The sojourn time of each user is generated in $(0, 400)$ seconds. This value could of course be increased, but here in order to reduce the learning time and be able to make many comparisons, we consider only small scenarios. Generated dataset contains separate 1501 sets, each set contains 1600 samples. Two sets are kept for validation, i.e., 3200 samples are used to evaluate models. The remaining containing 1499 sets are used for training. At each learning epoch, the model will go through 1600 samples (that is, input-output pairs). The transmission rate in each small slot is generated randomly between 0 and $5\delta/\Delta$. The rate function we use for SUMO scenarios (see Section 5.5) is

$$r(x) = \eta \left(1 + \kappa\, e^{-d(x,BS)/\sigma}\right), \tag{1}$$

where $d(x, BS)$ is the distance from position x to the BS, and η represents the noise level. For the SUMO scenarios in Section 5.5, we use $\kappa = 3$, $\sigma = 100$ and $\eta \sim \text{Uniform}(0.7, 1.3)$. The others parameters are equal to $J = 10, \Delta = 1$ s, $\delta = 2$ ms.

We remark that the rate function has an exponential decay with distance. For other smooth decay functions such as inverse square, the fundamental nature of the results (continuity, e.g.) and the conclusions (DFNN generate good quality solutions in a much shorter time compared to optimization solvers) of this paper will remain valid.

5.2. Comparison of different DFNN architectures

In this section, we will consider four different architectures of the DFNN model and compare their performances. The four models are as follows:

- **Model 1** - The first model used in this section contains 2 layers which are 1 hidden layer and 1 output layer. The hidden layer contains 500 units, and in total the model has $67,510$ parameters. The architecture of this model is illustrated in Figure 8.
- **Model 2** - As the first model, the second model contains 2 layers: 1 hidden layer and 1 output layer. However, the hidden layer contains 1000 units, and in total the model has $135,010$ parameters. We take the same number of layers as in Model 1 (but more units in hidden layers) in order to compare whether it is better to have more parameters.
- **Model 3** - As the two previous models, the third model contains 2 layers (1 hidden layer and 1 output layer). The hidden layer contains 100 units, and we have $13,510$ parameters in total. We take the same number of layers as in Model 1 (but fewer units in hidden layers) to compare whether it is better to have fewer parameters.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 120)	0
dense_1 (Dense)	(None, 500)	60500
activation_1 (Activation)	(None, 500)	0
batch_normalization_1 (Batch	(None, 500)	2000
dense_2 (Dense)	(None, 10)	5010
activation_2 (Activation)	(None, 10)	0

Total params: 67,510
Trainable params: 66,510
Non-trainable params: 1,000

Figure 8: Model 1 architecture.

- **Model 4** - The last model contains 10 layers which are 9 hidden layers and 1 output layer. Each hidden layer contains 82 units, and in total, the model contains $67,496$ parameters. We take a model that has almost the same number of parameters as Model 1, to compare whether it is better to have more layers or fewer layers.

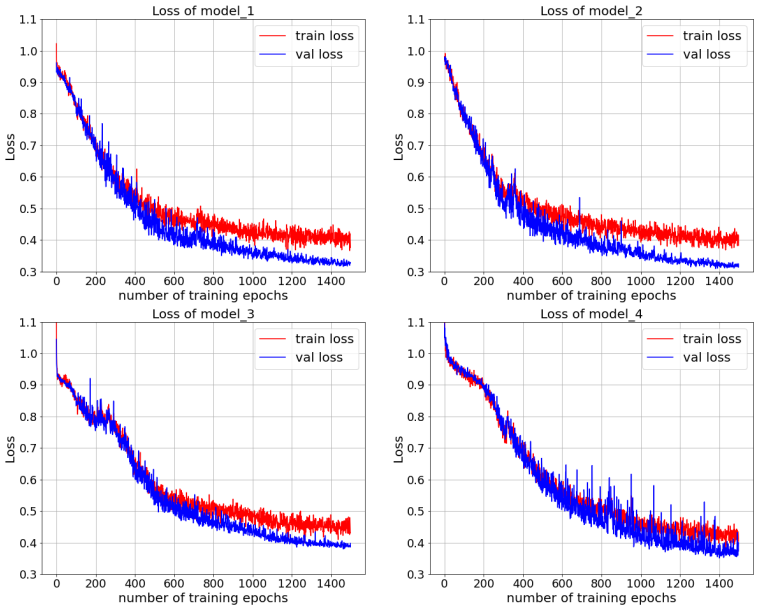
For the 4 models, the activation function used in hidden layers is the ReLU function, whereas the output layer uses the softmax function since we want the sum of the allocations to be equal to 1. In this comparison, we use the same loss function for all models, this loss function is denoted by bce_dice loss which is the sum of binary cross-entropy [25] and dice loss (which equals $1-$ dice coefficient [26]).

Remark 7 (Joint power control and channel allocation). *For the joint power control and channel allocation problem, we still compare the four above models except that the output layer of each model will be modified since it includes not only the channel allocation but also the power.*

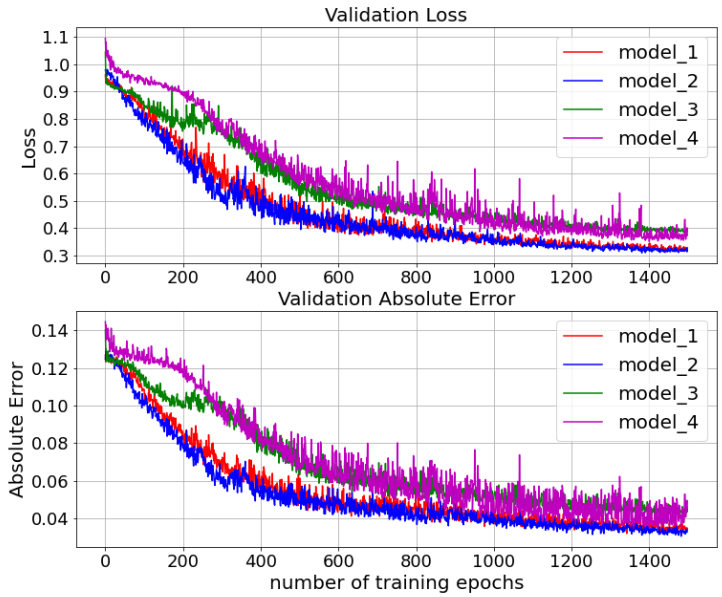
Figure 9a illustrates the loss of the 4 models on training and validation data. Figure 9b plots loss and absolute error of the 4 models on the same axis on validation set. The same quantities but for the problem of joint power control and channel allocation are shown in Figure 10a and Figure 10b respectively.

From these figures, we observe that for the model without power control:

- Having almost the same number of parameters, Model 1 with fewer layers is better than Model 4.
- Having the same layers, Model 1 and Model 2 with more parameters are better than Model 3.

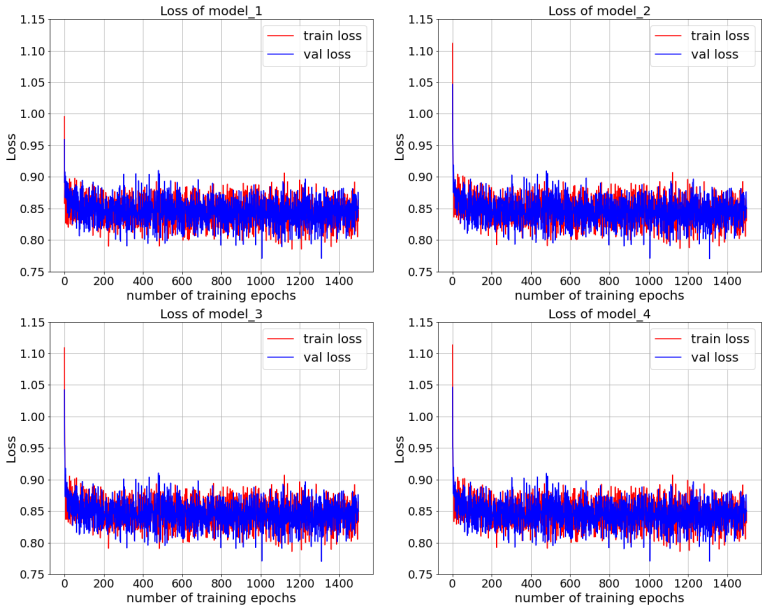


(a) Loss on training set and validation set of each model

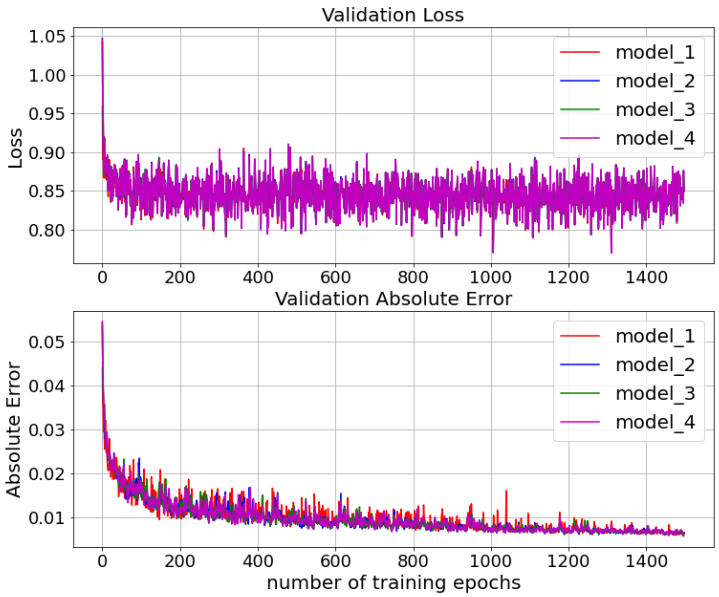


(b) Plot on same axis for loss and absolute error on validation set of all the four models

Figure 9: Comparison of the 4 DFNN models.



(a) Loss on training set and validation set of each model



(b) Plot on same axis for loss and absolute error on validation set of all the four models

Figure 10: Comparison of the 4 DFNN models for the joint power control and channel allocation problem.

- Model 1 and Model 2 behave similarly and have the same number of layers. However Model 1 has less parameters than Model 2 so it is less costly from a computational point of view. Therefore from now on we shall use Model 1 for other comparisons in the sequel.

For the model with power control:

- Having almost the same number of parameters, Model 1 with few layers is slightly better than Model 4, but the difference is quite small in this case.
- Having the same layers, models 1, 2 and 3 are almost the same but Model 3 has fewer parameters.

5.3. Comparisons of different loss functions

To compare the quality of the learning model obtained using different loss functions, we use the same model, that is Model 1. Figure 11 presents the results obtained with the Huber loss [27] against those obtained with bce_dice loss. So for the next comparisons, we shall use Model 1 and bce_dice loss.

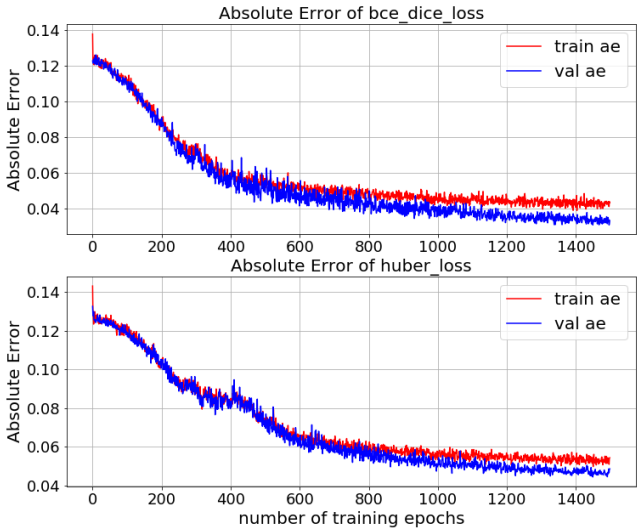
5.4. Comparison of different ordering schemes

For reasons mentioned above, we shall use Model 1 and bce_dice loss for this comparison. We compare the results obtained with 4 different ordering schemes: no order, current rate order, PF order and $(PS)^2S$ order [4]. Figure 12 presents our numerical results. The $(PS)^2S$ order seems the best one among the 4 ordering schemes.

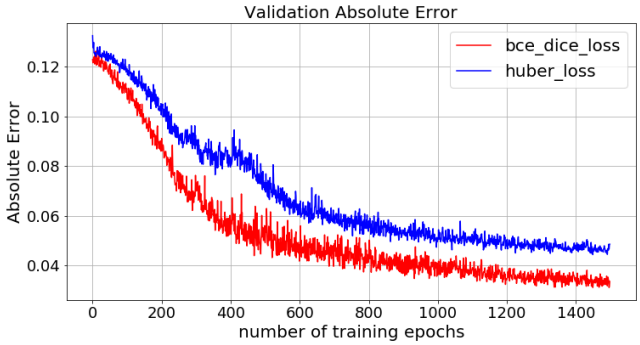
5.5. Comparison of learning the channel allocation against learning dual values

Next, we compare the approach which learns the dual values against the approach which learns directly the allocation. As before, for learning the allocation, we use Model 1 and bce_dice loss. Since the dual values need not lie in $[0, 1]$, we cannot use a softmax activation function in the output layer and bce_dice loss. We shall normalize the dual values by dividing each element by a fixed constant to reduce its magnitude before fitting it into the DFNN model and shall use Huber loss together with relu activation. The absolute errors of the different learning schemes are shown in Figure 13.

Since the values of the allocation and the dual are different, it is not possible to compare the losses or absolute errors directly as before. We shall instead compare three different learning schemes (learning allocation without order, learning allocation with $(PS)^2S$ order and learning the dual values) on two different mobility scenarios generated with the SUMO simulation package [10]. SUMO is an open source software designed for simulating mobility of moving users (vehicles, bus, truck, bicycle, pedestrian, ...) in large road traffic networks. It allows to import maps of different cities and simulate realistic mobility traces. SUMO is used to simulate the mobility of vehicular users in several specific regions of the Toulouse city and to compare our learning-based approaches against existing algorithms on realistic scenarios. The comparison proceeds in two steps: SUMO

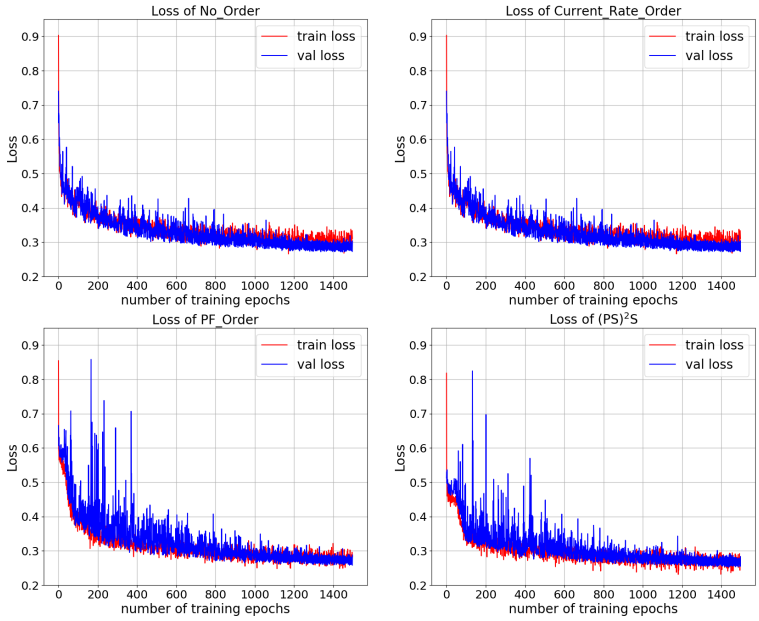


(a) Absolute error on training and validation set of the two losses

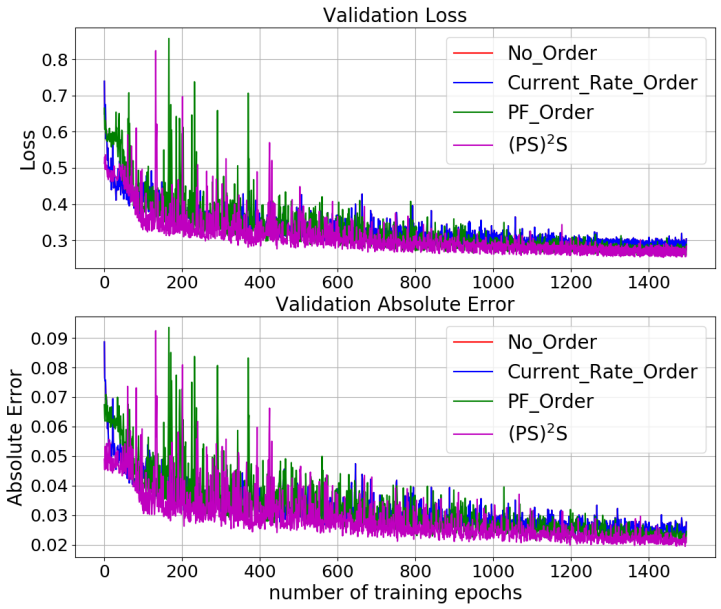


(b) Plot on same axis of the absolute error on validation of the two losses

Figure 11: Comparison of Loss functions.



(a) Loss on training and validation set



(b) Plot on same axis of the loss and absolute error on validation set

Figure 12: Comparison of 4 different types of ordering schemes.

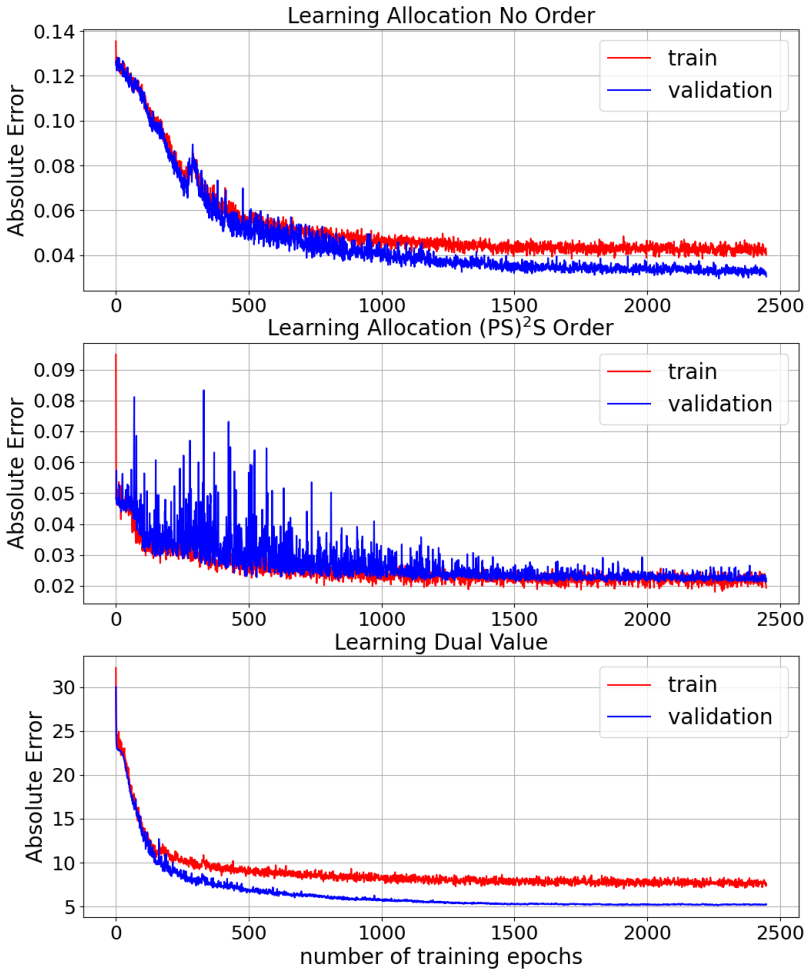


Figure 13: Comparisons of Absolute Error of the Learning Allocation and Learning Dual Value.

Figure 14: The Carmes borough in Toulouse, with one BS (Free Mobile type LTE1800). The actual size is $200m \times 400m$.

is first used for generating the mobility traces of vehicles, and then these traces are fed to a Python script which implements the algorithms and computes the value of the objective function for those algorithms.

The first scenario contains 244 users and lasts 61.7 minutes. The map of this scenario is shown in Figure 14. The results obtained with the different learning schemes on this scenario are shown in Figure 16, remark that by using the term "Learning Allocation No Order" in the figure, we means learning the primal value of the optimization which is the original allocation, without any order. The second scenario contains 214 users and lasts 62.4 minutes. The map of this scenario is shown in Figure 15. The data for BS location can be found on the website[4] of the French Frequency Agency (ANFR), which manages all radio frequencies in France. The results obtained with the different learning schemes on this scenario are shown in Figure 17. We also simulate two existing algorithms, $(PS)^2S$ [4] and PF [28] (which are also used in [1] for comparisons) in order to show that the approximation algorithm performs better than the existing algorithms. As mentioned above, for the learning algorithm, we use Model 1 and bce_dice loss.

When the number of learning epochs is large enough, the learning-based scheme performs well compared to STO1 and other algorithms. Learning the dual values outperforms the two other learning schemes: it is more stable, provides a better allocation and it converges faster than learning directly the allocation without any ordering scheme. It costs the same time for *prediction* as the two other approaches since they are using the same architecture (that is, Model 1). However learning directly allocation with a user ordering scheme requires to compute in addition the order, which consumes more computing time than the other two approaches.

[4]https://data.anfr.fr/anfr/portail

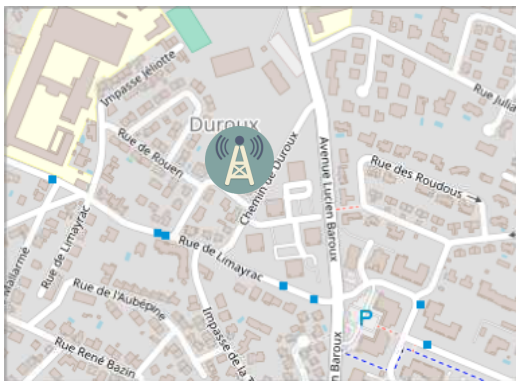


Figure 15: Duroux, one BS type LTE1800, operator SFR. The actual size is around $350 \times 500m$.

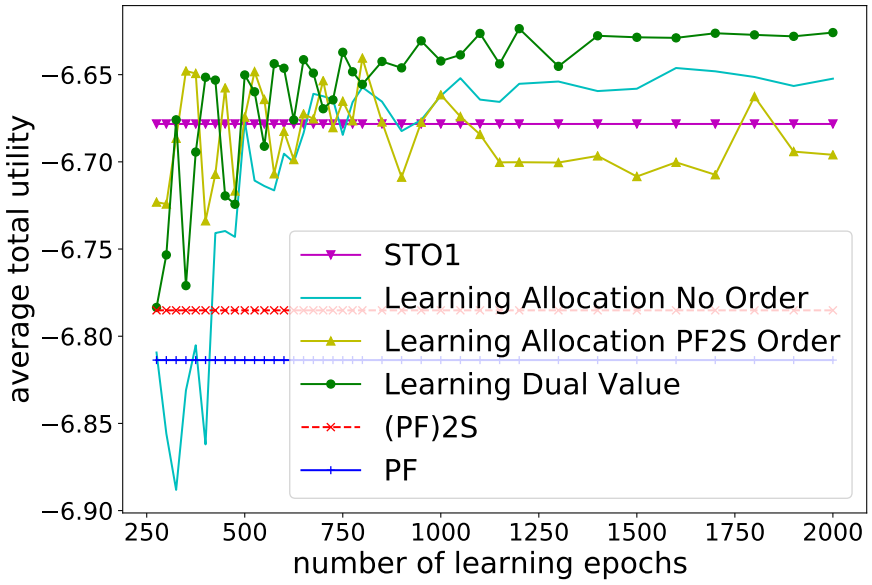


Figure 16: Comparisons of evaluated on Carmes scenario created by SUMO of Allocation and Dual Value.

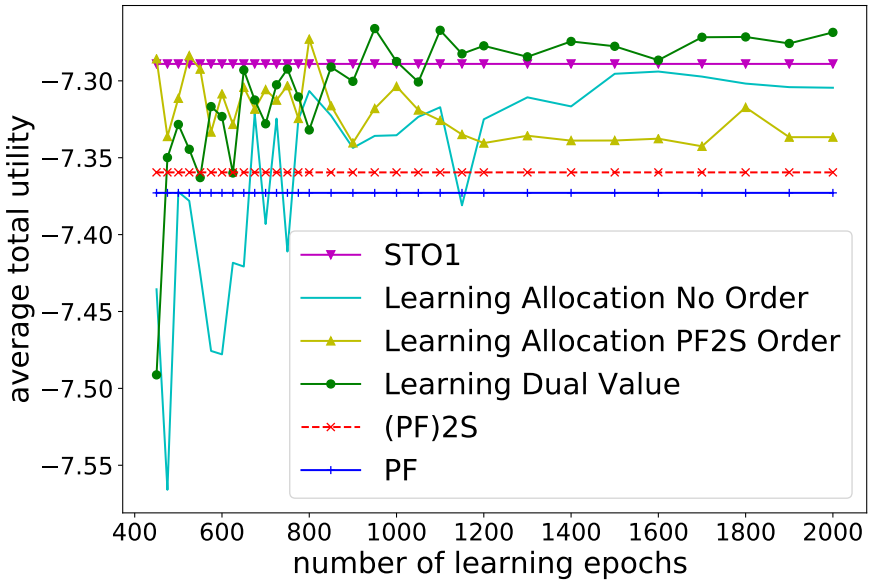


Figure 17: Comparisons of evaluated on Duroux scenario created by SUMO of Allocation and Dual Value.

5.6. Influence of the number of vehicles

We now investigate the influence of the dimension of the input problems on the utility as well the prediction times of the proposed DFNN approach and compare them with that of the directly solving with Mosek.

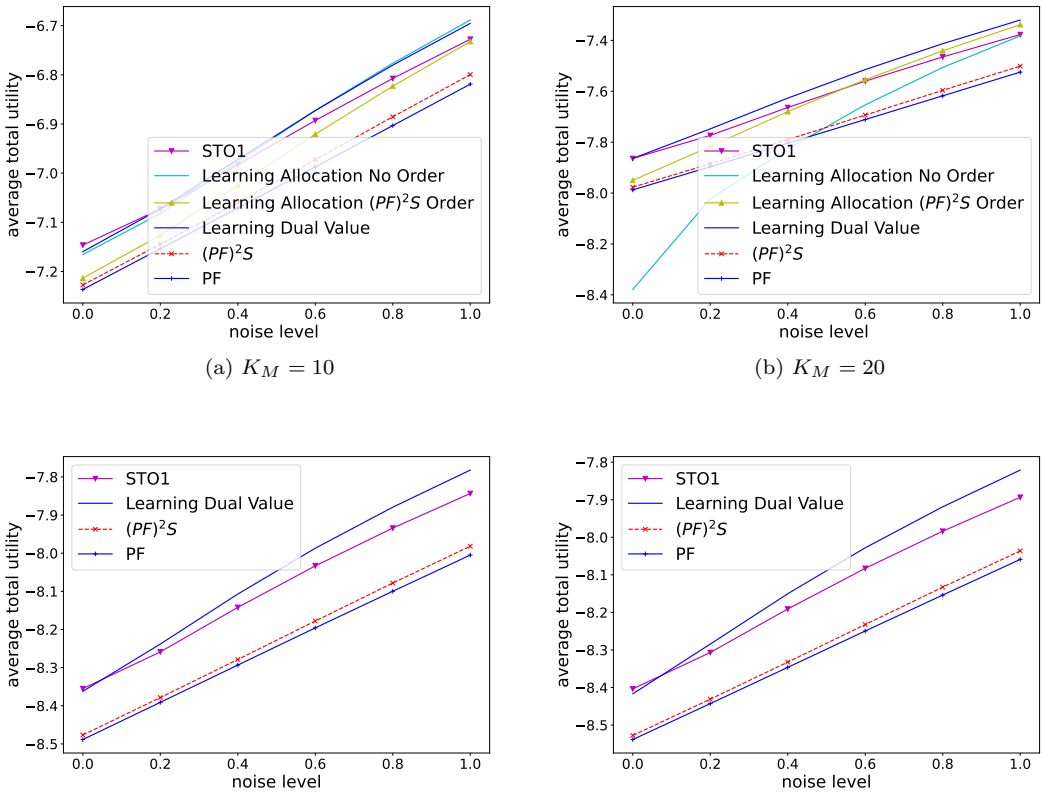
For this, in the Carmes borough, we increase the arrival rate of users so as to have more users in the network, which corresponds to a higher K_M. Five different arrivals rates were chosen corresponding to $K_M = 10, 20, 30, 40, 50$. When K_M increases, the size of input, output, and the complexity of the optimization problem inside STO1 also increase. That is the reason why we need to increase the size of DFNN architecture in order to increase the quality of approximation. In more detail, we increase the size of DFNN architecture by increasing the number of neurons in the hidden layer while keeping the same unique hidden layer. To explain why we do that, we refer readers to this reference [29] in which the author suggests how to prune the number of hidden layers and the number of neurons in the hidden layer. For learning Dual value, table 1 shows the corresponding the number of neurons in the unique hidden layer, the corresponding number of epochs at which the validation loss converges, the corresponding training time (in seconds) for each epoch and the corresponding whole training time (which equals the number of epochs times the training time in each epoch) for different values of K_M. The weights chosen to evaluate for SUMO scenarios are taken at the epoch when the validation loss converges, i.e, there is no clearly seen improvement.

K_M	# neurons in hidden layer	time training per epoch	# epochs to converge	whole training time
$K_M = 10$	500	0.265	1400	371.0
$K_M = 20$	700	0.282	1800	507.0
$K_M = 30$	1000	0.297	3000	891.0
$K_M = 40$	3000	0.301	4500	1354.5
$K_M = 50$	10000	0.393	7000	2751.0

Table 1: Training time in second.

The average utilities obtained by the different heuristics for these values of $K_M = 10$ are shown in figure 18 as a function of the noise level, η. It is observed that Learning with dual values outperforms other heuristics in terms of the average utilities for all noise levels except level 0.0 where STO1 is marginally better. We also observe that learning without order (that is, learning directly the output of the solver) does not scale well when K_M increases from 10 to 20. For this reason, in the experiments for $K_M = 40$ and $K_M = 50$, only the Learning from dual values heuristic is presented. For $K_M = 30$, a similar result was observed. Its plot has been omitted for conciseness.

For learning the primal value (the original allocation) when K_M is larger, we observed that it takes very long training epochs for the loss function to converge and the convergence might be far from the true value since the convergent loss is still very high. For example, for $K_M = 20$, learning dual value needs 1800 epochs



(a) $K_M = 10$ (b) $K_M = 20$

(c) $K_M = 40$ (d) $K_M = 50$

Figure 18: Comparision of the average utility with the noise level for different values of K_M.

to converge as shown in table 1, but learning primal value needs more than 9000 epochs to converge, we stopped training at epoch 9000 and observed average absolute error around 0.04, i.e, the total absolute error is around $0.04 * 20 = 0.8$ while the maximum absolute error is equal to 2 where the prediction vector and the true vector (which has unique position equal to 1 and 0 everywhere else) totally disagree. That is why the performance is bad as shown in figure 5.6.

5.7. Computing times

In this time, we use another machine to simulate the system, since the old machine (the one we used in 2020 to have the results in the previous version of this paper) does not work anymore. To make a consistent measurement, we decide to redo all experiments with another machine, those measurements are shown in the below paragraph and table 2.

The computing time of STO1 depends on the convex optimization solver

used, whereas the learning algorithm has only to feed the DFNN model with the input matrix. We consider the same setting as in Section 5.2, that is $K = 10$ (there are exactly 10 users in the system)[5] and the short term horizon is $J = 10$ seconds. For these values, the average computing time of Mosek is around 20.61 ms, whereas the prediction with the DFNN model (1 hidden layer with 500 neurons) takes only 0.92 ms on average. When adding power control, Mosek solves the optimization problem in around 70.39 ms, while the prediction of the DFNN model (almost the same with Model 1 but the output layer contains power vector in addition, which contains 73,020 parameters) takes 1.04 ms on average. These computing times are averaged over 3000 samples. The experiments were done on a a AMD 7413 24-core processor.

Note that, for STO1 (which uses Mosek to solve optimization) the computing time increases when the number of users in the system increases. For example, when K varies from 1 to 10, Mosek computing time varies from 19.94 milliseconds ($K = 1$) to 20.61 milliseconds ($K = 10$). For DFNN, the computing time changes only when we change the its architecture, that is only when K_M changes. For 5 different values of K_M, Table 2 shows detailed comparisons between computing time (in ms) of STO1 versus DFNN. The architecture DFNN for a given value of K_M is the one presented in Sec. 5.6 and Table 1. The entries of the table should be read as follows. For example, for $K_M = 10$, the computting time of STO1 varies from 19.94 ms to 20.56 ms when K varies from 1 to 10. The inference time of the DFNN depends only on K_M and hence has only one value which is 0.92 ms.

K_M	10	20	30	40	50
K	$[1, 10]$	$[11, 20]$	$[21, 30]$	$[31, 40]$	$[41, 50]$
STO1	$[19.9, 20.6]$	$[20.6, 22]$	$[22, 22.6]$	$[22.5, 23.2]$	$[23.3, 23.8]$
DFNN	0.92	0.93	0.95	1.04	1.18

Table 2: Comparison of computing time (in milliseconds) between STO1 and DFNN for different numbers of users K and K_M.

From the above measurements, we can conclude that even running with CPU, we still can see the improvement of DFNN compared to STO1.

6. Summary and Discussion

We have proposed to use a DFNN for learning the channel allocation obtained with one of the heuristics (STO1) introduced in [1] and [2]. Numerical results on SUMO scenarios show that the learning-based method yields approximate yet satisfactory channel allocations with much less computation time as

[5]Recall that K is the number of cars that are present at a given time, whereas K_M is the maximum number of cars that are simultaneously present in the coverage range. That is $K \leq K_M$.

long as there are enough learning epochs. The state of the DFNN is defined in such a way that the model is not restricted to a particular scenario, that is, it can learn the channel allocation for a general network.

There are several directions of research that can be investigated to improve the learning algorithm, such as creating a better generator of data for learning, choosing a better loss function, choosing a better architecture of the DFNN model in which how to prune architecture (number of hidden layers and number of neurons for each layer) for different K_M in an automatic way would be one of the most important things, and other things such as choosing a better optimizer, the learning rate, etc.

Acknowledgment

The authors would like to thank Continental Digital Service France for supporting partially this work.

References

[1] N. Nguyen, O. Brun, B. Prabhu, An algorithm for improved proportional-fair utility for vehicular users, The 25th International Conference on Analytical and Stochastic Modelling Techniques and Applications ASMTA-2019 (May 2019).

[2] N. Nguyen, O. Brun, B. Prabhu, Joint downlink power control and channel allocation based on a partial view of future channel conditions, The 15th Workshop on Resource Allocation, Cooperation and Competition in Wireless Networks (June 2020).

[3] T. T. N. Nguyen, O. Brun, B. J. Prabhu, Learning resource allocation algorithms for cellular networks, in: É. Renault, S. Boumerdassi, P. Mühlethaler (Eds.), Machine Learning for Networking - Third International Conference, MLN 2020, Paris, France, November 24-26, 2020, Vol. 12629 of Lecture Notes in Computer Science, Springer, 2020, pp. 184–203.

[4] R. Margolies, A. Sridharan, V. Aggarwal, R. Jana, N. K. Shankaranarayanan, V. A. Vaishampayan, G. Zussman, Exploiting mobility in proportional fair cellular scheduling: Measurements and algorithms, IEEE/ACM Trans. Netw. 24 (1) (2016) 355–367.

[5] D. Tse, P. Viswanath, Fundamentals of Wireless Communication, Cambridge University Press, 2005.

[6] Y. Yi, M. Chiang, Stochastic network utility maximisation – a tribute to Kelly's paper published in this journal a decade ago, European Transactions on Telecommunications 19 (4) (2008) 421–442.

[7] H. J. Kushner, P. A. Whiting, Convergence of proportional-fair sharing algorithms under general conditions, IEEE Transactions on Wireless Communications 3 (4) (2004) 1250–1259.

[8] S. Diamond, S. Boyd, CVXPY: A Python-embedded modeling language for convex optimization, Journal of Machine Learning Research 17 (83) (2016) 1–5.

[9] MOSEK ApS, MOSEK Optimizer API for Python manual. Version 9.2.21 (2019).
URL `https://docs.mosek.com/9.2/pythonapi/index.html`

[10] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, E. Wießner, Microscopic traffic simulation using sumo, in: The 21st IEEE International Conference on Intelligent Transportation Systems, IEEE, 2018.

[11] A. Kolmogorov, On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition, Dokl. Akad. Nauk SSSR 114 (5) (1957) 953–956.

[12] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals and Systems 2 (4) (1989) 303–314.

[13] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (5) (1989) 359 – 366.

[14] H. Montanelli, H. Yang, Error bounds for deep ReLU networks using the kolmogorov–arnold superposition theorem (2019). arXiv:1906.11945.

[15] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Networks 4 (2) (1991) 251 – 257.

[16] K. Gregor, Y. LeCun, Learning fast approximations of sparse coding, in: Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10, Omnipress, Madison, WI, USA, 2010, p. 399–406.

[17] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, N. D. Sidiropoulos, Learning to optimize: Training deep neural networks for wireless resource management, CoRR abs/1705.09412 (2017).

[18] J. Huang, V. Subramanian, R. Agrawal, R. Berry, Downlink scheduling and resource allocation for ofdm systems, IEEE Transactions on Wireless Communications 8 (2009) 288–296.

[19] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, `http://www.deeplearningbook.org`.

[20] S. Sun, W. Chen, L. Wang, X. Liu, T.-Y. Liu, On the depth of deep neural networks: A theoretical view, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, 2016, p. 2066–2072.

[21] Y. A. LeCun, L. Bottou, G. B. Orr, K.-R. Müller, Efficient backprop, in: G. Montavon, G. B. Orr, K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade: Second Edition, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 9–48.

[22] D. Kingma, J. Ba, Adam: A method for stochastic optimization, International Conference on Learning Representations (12 2014).

[23] S. J. Reddi, S. Kale, S. Kumar, On the convergence of adam and beyond, CoRR abs/1904.09237 (2019).

[24] F. Chollet, et al., Keras, `https://github.com/fchollet/keras` (2015).

[25] Wikipedia, Cross entropy`https://en.wikipedia.org/wiki/Cross_entropy` (20 June 2020).

[26] Wikipedia, Sorensen dice coefficient`https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient` (28 July 2020).

[27] Wikipedia, Huber loss`https://en.wikipedia.org/wiki/Huber_loss` (29 May 2020).

[28] F. Kelly, Charging and rate control for elastic traffic, European Transactions on Telecommunications 8 (1) (1997) 33–37.

[29] J. Heaton, Artificial Intelligence for Humans: Deep learning and neural networks, Artificial Intelligence for Humans, Heaton Research, Incorporated., 2015.
URL `https://books.google.fr/books?id=q9mijgEACAAJ`

Appendix A. Proof of Proposition 1

Before proving Proposition 1, we need a lemma, in which we shall consider the following equivalent problem:

$$\begin{cases} \text{maximize} \sum_{i=1}^{K(t)} \log\left(C_i\right) \\ \mathbf{C} \in \mathcal{C} \end{cases}, \tag{II_R}$$

where

$$\mathcal{C} = \{\mathbf{C} = (C_1, \ldots, C_K) \mid C_i = \sum_{j=t}^{t+J} \alpha_{ij} r_{ij} + c_i \, \forall i, \sum_i \alpha_{ij} = 1 \, \forall j, \alpha_{ij} \geq 0 \, \forall i, j\}.$$

Then (II_R) is equivalent to (I_R), and $\mathcal{C}$ is a convex set. Let $\mathbf{C}^*$ be the optimal solution of (II_R).

Lemma 1. *The optimal solution $\mathbf{C}^*$ is uniquely defined by R and continuous in R.*

Proof. Indeed since (II_R) is a convex optimization problem, we have for any $\mathbf{C} \in \mathcal{C}$:

$$\nabla O_{\mathbf{C}}(\mathbf{C}^*)(\mathbf{C} - \mathbf{C}^*)^T \leq 0. \tag{A.1}$$

where $O(\mathbf{C}) = \sum_i \log(C_i)$. Let us take R' close to R, with $|R' - R|_\infty = \epsilon$ and assume that $\mathbf{C}'^*$ is solution corresponding to the matrix R'. By definition of the set $\mathcal{C}$, there exists α^* and α'^* such that:

$$\begin{aligned} C_i^* &= \sum_{j=t}^{t+J} \alpha_{ij}^* r_{ij} + c_i, \forall i, \\ C_i'^* &= \sum_{j=t}^{t+J} \alpha_{ij}'^* r'_{ij} + c'_i, \forall i. \end{aligned} \tag{A.2}$$

It is obvious that $C_i^*, C_i'^*$ are strictly positive $\forall i$. From (A.1) we obtain

$$\sum_{i=1}^{K} \frac{C_i}{C_i^*} \leq K \quad \forall \mathbf{C} = (C_1, C_2, ...C_K) \in \mathcal{C}. \tag{A.3}$$

It follows that

$$\begin{aligned} \left| \sum_{i=1}^{K} \frac{C_i'^*}{C_i^*} \right| &= \left| \sum_{i=1}^{K} \frac{\sum_{j=t}^{t+J} \alpha_{ij}'^* r'_{ij} + c'_i}{\sum_{j=t}^{t+J} \alpha_{ij}^* r_{ij} + c_i} \right| \\ &= \left| \sum_{i=1}^{K} \frac{\left(\sum_j \alpha_{ij}'^* r_{ij} + c_i\right) + \left(\sum_{j=t}^{t+J} \alpha_{ij}'^* (r'_{ij} - r_{ij}) + (c'_i - c_i)\right)}{\sum_{j=t}^{t+J} \alpha_{ij}^* r_{ij} + c_i} \right| \\ &\leq \left| \sum_{i=1}^{K} \frac{C_i}{C_i^*} \right| + \left| \sum_{i=1}^{K} \frac{\left(\sum_{j=t}^{t+J} \alpha_{ij}'^* (r'_{ij} - r_{ij}) + (c'_i - c_i)\right)}{\sum_{j=t}^{t+J} \alpha_{ij}^* r_{ij} + c_i} \right| \\ &\leq \left| \sum_{i=1}^{K} \frac{C_i}{C_i^*} \right| + A_1 \epsilon \\ &\leq K + A_1 \epsilon, \end{aligned} \tag{A.4}$$

where $C_i = \sum_j \alpha_{ij}'^* r_{ij} + c_i$ for all i and $A_1 = \sum_{i=1}^{K} \frac{(J+1)+1}{C_i^*}$ is a positive number which does not depend on ϵ.

The fourth implication follows from $|R' - R| = \epsilon$ and $0 \leq \alpha_{ij}'^* \leq 1$, whereas the last implication follows from (A.3) and the fact that $\mathbf{C} = (C_i)_i \in \mathcal{C}$. So from

(A.4) we get $\sum_{i=1}^{K} \frac{C_i'^*}{C_i^*} \leq K + A_1\epsilon$. Similarly, we have $\sum_{i=1}^{K} \frac{C_i^*}{C_i'^*} \leq K + A_2\epsilon$, where A_2 is also a positive number not depending on ϵ. It implies that

$$\sum_{i=1}^{K} \left(\frac{C_i^*}{C_i'^*} + \frac{C_i'^*}{C_i^*} \right) \leq 2K + (A_1 + A_2)\epsilon.$$

Combining this with the fact that for each i we have $\left(\frac{C_i^*}{C_i'^*} + \frac{C_i'^*}{C_i^*} \right) \geq 2$ (this is implied by the AM–GM inequality which claims that $a + b \geq \sqrt{ab}$ for all $a \geq 0$ and $b \geq 0$), we obtain

$$\left(\frac{C_i^*}{C_i'^*} + \frac{C_i'^*}{C_i^*} \right) \leq 2 + (A_1 + A_2)\epsilon, \ \forall\, i.$$

Define $z_i = \frac{C_i^*}{C_i'^*}$. Then, we have $z_i > 0$ and $z_i + 1/z_i \leq 2 + (A_1 + A_2)\epsilon$. This is a quadratic inequality, and solving it we get

$$1 + D\epsilon - \sqrt{(D\epsilon)^2 + 2D\epsilon} \ \leq \ z_i \ \leq \ 1 + D\epsilon + \sqrt{(D\epsilon)^2 + 2D\epsilon}, \tag{A.5}$$

for all i, where $D = \frac{A_1 + A_2}{2} > 0$ does not depend on ϵ.

From (A.4) and (A.5) we have that:

- From (A.4), if we take $R' = R$, the last inequality becomes $\sum_{i=1}^{K} \frac{C_i'^*}{C_i^*} \leq K$, and also $\sum_{i=1}^{K} \frac{C_i^*}{C_i'^*} \leq K$. Similarly to above implications, this implies $z_i + 1/z_i = 2$ for all i. It implies that $t_i = 1$ for all i, i.e, $C_i^* = C_i'^*$ for all i, which shows the uniqueness of the solution of (II_R).
- From (A.5), we obtain that

$$\lim_{\epsilon \to 0} \frac{C_i'^*}{C_i^*} = 1 \quad \forall i, \tag{A.6}$$

  i.e., $\lim_{\epsilon \to 0} C_i'^* = C_i$ for all i, which proves the continuity of $\mathbf{C}^*$ in R.

■

Proof. (proof of proposition 1) Recall that $\mathcal{A} = \{R | \# I_t(R) = 1\}$, i.e., given $R \in \mathcal{A}$, there exists an unique i_0 such that $\rho_{i_0 t} = 0$ and for every $i \neq i_0$ we have $\rho_{it} > 0$. It yields

$$\frac{r_{i_0 t}}{C_{i_0}^*} = \lambda_t^* > \lambda_t^* - \rho_{it}^* = \frac{r_{i_t}}{C_i^*}, \quad \forall i \neq i_0, \tag{A.7}$$

and $\alpha_{i_0 t}^* = 1$, whereas $\alpha_{it}^* = 0$ for all $i \neq i_0$. This means that the allocation $\alpha_{\mathbf{t}}^*$ is unique for every input state R in $\mathcal{A}$.

Consider $R \in \mathcal{A}$ and R' close to R: $|R' - R|_\infty = \epsilon$. Since we consider the continuity of the current allocation only, our objective is to prove that $\alpha_{\mathbf{t}}'^* = (\alpha_{it}'^*)_i$ is close to $\alpha_{\mathbf{t}}^* = (\alpha_{it}^*)_i$. In fact we have that $\alpha_{\mathbf{t}}'^* = \alpha_{\mathbf{t}}^*$ if ϵ is

small enough. Indeed, from claim (A.5) in Lemma 1, we have $\frac{C_i'^*}{C_i^*} = 1 + \mathcal{O}(\sqrt{\epsilon})$ $(0 < \lim_{\epsilon \to 0} \frac{\mathcal{O}(\sqrt{\epsilon})}{\epsilon} < +\infty)$, where $(C_i^*)_i$ and $(C_i'^*)_i$ are the solutions of (II_R) corresponding to inputs R and R', respectively. Since $R \in \mathcal{A}$, there exists i_0 such that $\frac{r_{i_0 t}}{C_{i_0}^*} > \frac{r_{i_t}}{C_i^*}$ for all $i \neq i_0$. It implies that $\frac{r'_{i_0 t}}{C_{i_0}'^*} > \frac{r'_{i_t}}{C_i'^*}$ for all $i \neq i_0$ if ϵ is small enough. In turn, this implies that $\alpha_{i_0 t}'^* = 1$ and $\alpha_{it}'^* = 0$ for all $i \neq i_0$, that is, $\alpha_{\mathbf{t}}^* = \alpha_{\mathbf{t}}'^*$. ■

Appendix B. Proof of Proposition 2

Let us characterize the set $\mathcal{B} = \{R | \# I_1(R) \geq 2\}$. The set $\mathcal{A}$ will be the complement. Since we have 2 users,

$$\rho_{11}^* = \rho_{21}^* = 0. \tag{B.1}$$

Combining with the fourth condition in (KKT) we have:

$$\frac{r_{11}}{C_1^*} = \frac{r_{21}}{C_2^*}. \tag{B.2}$$

There are three possible cases:

- Case 1: $\alpha_{12}^* \in (0,1)$, which implies that $\alpha_{22}^* \in (0,1)$.
- Case 2: $\alpha_{12}^* = 0$, which implies that $\alpha_{22}^* = 1$.
- Case 3: $\alpha_{12}^* = 1$, which implies that $\alpha_{22}^* = 0$.

Case 1: $\alpha_{12}^* \in (0,1)$ and therefore $\alpha_{22}^* \in (0,1)$. From the second condition in (KKT), it implies $\rho_{12}^* = \rho_{22}^* = 0$. Combining with the fourth condition in (KKT) we have:

$$\frac{r_{12}}{C_1^*} = \frac{r_{22}}{C_2^*}. \tag{B.3}$$

Dividing (B.3) by (B.2) side by side we get:

$$\frac{r_{12}}{r_{11}} = \frac{r_{22}}{r_{21}}. \tag{B.4}$$

Conversely, if $\frac{r_{12}}{r_{11}} = \frac{r_{22}}{r_{21}}$, we shall prove that the solution of (I_R) is of the form

$$\begin{bmatrix} \alpha_{11}^* & \alpha_{21}^* \\ \alpha_{12}^* & \alpha_{22}^* \end{bmatrix} = \begin{bmatrix} a & 1-a \\ \frac{1+t}{2t} - \frac{a}{t} & \frac{t-1}{2t} + \frac{a}{t} \end{bmatrix},$$

for any $a \in [\max(0, \frac{1-t}{2}), \min(1, \frac{1+t}{2})]$, where $t = \frac{r_{12}}{r_{11}} = \frac{r_{22}}{r_{21}}$. Indeed, problem (I_R) is equivalent to the following problem:

$$\begin{aligned} & \max \left[\log(\alpha_{11} + t\alpha_{12}) + \log(\alpha_{21} + t\alpha_{22})\right] \\ \Leftrightarrow & \max \left[\log(\alpha_{11} + t\alpha_{12}) + \log((1 - \alpha_{11}) + t(1 - \alpha_{12}))\right] \\ \Leftrightarrow & \max \left[\log(X) + \log\left((1+t) - X\right)\right] (\text{where } X = \alpha_{11} + t\alpha_{12}) \\ \Leftrightarrow & \max \left[X\left((1+t) - X\right)\right]. \end{aligned} \tag{B.5}$$

We have

$$X\left((1+t)-X\right) \leq \frac{(X+((1+t)-X))^2}{4} = \frac{(1+t)^2}{4},$$

and the equality occurs if only if $X = \frac{1+t}{2}$, i.e, if $\alpha_{12} = \frac{1+t}{2t} - \frac{\alpha_{11}}{t}$. So, we get

$$\alpha^* = \begin{bmatrix} a & 1-a \\ \frac{1+t}{2t} - \frac{a}{t} & \frac{t-1}{2t} + \frac{a}{t} \end{bmatrix},$$

with $a \in [\max(0, \frac{1-t}{2}), \min(1, \frac{1+t}{2})]$ to guarantee that every value in the matrix is in $[0, 1]$.
Conclusion for case 1: on the line $\frac{r_{12}}{r_{11}} = \frac{r_{22}}{r_{21}}$, the solution is not unique, the solutions are of the above form.

Case 2: $\alpha_{12}^* = 0$ and therefore $\alpha_{22}^* = 1$. This implies

$$\rho_{22}^* = 0.$$

Therefore:

$$\frac{r_{22}}{C_2^*} = \lambda_2^* \geq \lambda_2^* - \rho_{12}^* = \frac{r_{12}}{C_1^*}. \tag{B.6}$$

Combining with (B.2), we get

$$\frac{r_{22}}{r_{21}} \geq \frac{r_{12}}{r_{11}}. \tag{B.7}$$

On the other hand, in this case Problem (I_R) amounts to finding the maximum of

$$[\log(r_{11}\alpha_{11}) + \log(r_{21}(1-\alpha_{11}) + r_{22})] := f(\alpha_{11}).$$

$$f'(\alpha_{11}) = 0 \Leftrightarrow \alpha_{11} = \frac{1}{2}\left(1 + \frac{r_{22}}{r_{21}}\right). \tag{B.8}$$

Let us check whether the optimal α_{11}^* is on the boundary or is the above stationary point. On the boundary we have:

$$f(0) = -\infty,$$

$$f(1) = \log(r_{11}) + \log(r_{22}) > f(0).$$

For the stationary point, we have two cases:

- If $\alpha_{11} = \frac{1}{2}\left(1 + \frac{r_{22}}{r_{21}}\right) \in [0, 1]$, i.e $r_{22} \leq r_{21}$, then we have:

$$f\left(\frac{1}{2}(1 + \frac{r_{22}}{r_{21}})\right) = \log\left(r_{11}\frac{r_{21}+r_{22}}{2r_{21}}\right) + \log\left(\frac{r_{21}+r_{22}}{2}\right) \tag{B.9}$$

$$\geq \log(r_{11}) + \log(r_{22}) (\text{since } r_{22} \leq r_{21}) \tag{B.10}$$

$$= f(1) \tag{B.11}$$

  So in this case the optimal solution is given by $\alpha_{11} = \frac{1}{2}\left(1 + \frac{r_{22}}{r_{21}}\right)$ and the matrix input has to satisfy $\frac{r_{22}}{r_{21}} \leq 1$.

- If $\alpha_{11} = \frac{1}{2}\left(1 + \frac{r_{22}}{r_{21}}\right) > 1$ then we get $\frac{r_{22}}{r_{21}} > 1$ and f attains its maximum at $\alpha_{11} = 1$ (and therefore $\alpha_{21} = 0$). But it can not happens in the set $\mathcal{B}$ with $\rho_{11}^* = \rho_{21}^* = 0$. Indeed, in this case, $C_1^* = r_{11}, C_2^* = r_{22}$, and

$$\lambda_1^* = \max\left(\frac{r_{11}}{C_1^*}, \frac{r_{21}}{C_2^*}\right) = \max\left(1, \frac{r_{21}}{r_{22}}\right) = 1. \tag{B.12}$$

  Combining with the fourth condition of (KKT), we get

$$\rho_{21}^* = \lambda_1^* - \frac{r_{21}}{C_2^*} = 1 - \frac{r_{21}}{r_{22}} > 0. \tag{B.13}$$

  That is a contradiction.

Conversely, if $\frac{r_{22}}{r_{21}} > \frac{r_{12}}{r_{11}}$ (here we consider only the strict inequality since the equality has been already considered in case 1) and $0 \leq \frac{r_{22}}{r_{21}} \leq 1$. Then the solution of (I_R) is this form:

$$\begin{bmatrix} \alpha_{11}^* & \alpha_{21}^* \\ \alpha_{12}^* & \alpha_{22}^* \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\left(1 + \frac{r_{22}}{r_{21}}\right) & \frac{1}{2}\left(1 - \frac{r_{22}}{r_{21}}\right) \\ 0 & 1 \end{bmatrix}.$$

Indeed, from $\frac{r_{22}}{r_{21}} > \frac{r_{12}}{r_{11}}$ and (B.2) we obtain that

$$\frac{r_{22}}{C_2^*} > \frac{r_{12}}{C_1^*} \implies \rho_{12}^* = \lambda_2^* - \frac{r_{12}}{C_1^*} > \lambda_2^* - \frac{r_{22}}{C_2^*} = \rho_{22}^* \geq 0. \tag{B.14}$$

So $\rho_{12}^* > 0 \xrightarrow{KKT} \alpha_{12}^* = 0$ and therefore $\alpha_{22}^* = 1$. Therefore, (I_R) becomes an optimization problem of one variable α_{11}, and by solving it we get $\alpha_{11}^* = \frac{1}{2}\left(1 + \frac{r_{22}}{r_{21}}\right) \in (\frac{1}{2}, 1]$ and therefore $\alpha_{21}^* = \frac{1}{2}\left(1 - \frac{r_{22}}{r_{21}}\right) \in [0, \frac{1}{2})$. From these values we can compute the dual value:

$$\frac{r_{11}}{C_1^*} = \frac{r_{21}}{C_2^*} = \frac{2r_{21}}{r_{21} + r_{22}} (\text{ therefore } = \lambda_1^*) \implies \rho_{11}^* = \rho_{21}^* = 0. \tag{B.15}$$

Conclusion for case 2: The region in this case is equal to the region defined by $\frac{r_{22}}{r_{21}} \geq \frac{r_{12}}{r_{11}}$ and $0 \leq \frac{r_{22}}{r_{21}} \leq 1$. On the triangle defined by $\frac{r_{22}}{r_{21}} > \frac{r_{12}}{r_{11}}$ (strictly inequality here) and $0 \leq \frac{r_{22}}{r_{21}} \leq 1$, the solution is unique, and is given by

$$\alpha_{11} = \frac{1}{2}\left(1 + \frac{r_{22}}{r_{21}}\right).$$

The formula shows the continuity on the interior of the triangle. In this region, except on the segment defined by $\frac{r_{22}}{r_{21}} = 1 \geq \frac{r_{12}}{r_{11}}$, the solution is not integer.

Case 3: Similar to case 2, if $\alpha_{12}^* = 1$, then R has to satisfy: $0 \leq \frac{r_{12}}{r_{11}} \leq 1$ and $\frac{r_{12}}{r_{11}} \geq \frac{r_{22}}{r_{21}}$. When $0 \leq \frac{r_{12}}{r_{11}} \leq 1$ and $\frac{r_{12}}{r_{11}} > \frac{r_{22}}{r_{21}}$, the solution is unique and given by

$$\begin{bmatrix} \alpha_{11}^* & \alpha_{21}^* \\ \alpha_{12}^* & \alpha_{22}^* \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\left(1 - \frac{r_{12}}{r_{11}}\right) & \frac{1}{2}\left(1 + \frac{r_{12}}{r_{11}}\right) \\ 1 & 0 \end{bmatrix}.$$

Conclusion for case 3: The region in this case is equal to the region defined by $\frac{r_{22}}{r_{21}} \leq \frac{r_{12}}{r_{11}}$ *and* $0 \leq \frac{r_{12}}{r_{11}} \leq 1$. *On the triangle defined by* $\frac{r_{22}}{r_{21}} < \frac{r_{12}}{r_{11}}$ *and* $0 \leq \frac{r_{12}}{r_{11}} \leq 1$, *the solution is unique, and is given by*

$$\alpha_{11} = \frac{1}{2}\left(1 - \frac{r_{12}}{r_{11}}\right). \tag{B.16}$$

The formula shows the continuity on the interior of the triangle. In this region, except on the segment $\frac{r_{12}}{r_{11}} = 1 \geq \frac{r_{22}}{r_{21}}$, *the solution is not integer.*

The above analysis characterizes the structure of the set $\mathcal{B}$. Since $\mathcal{A}$ is the complement of $\mathcal{B}$, it contains two connected components: one defined by $\frac{r_{12}}{r_{11}} > 1$ and $\frac{r_{12}}{r_{11}} > \frac{r_{22}}{r_{21}}$; and the other one defined by $\frac{r_{22}}{r_{21}} > 1$ and $\frac{r_{12}}{r_{11}} < \frac{r_{22}}{r_{21}}$.

Recall that on $\mathcal{A}$ we have: $\#I_1(R) = 1$, i.e at time slot 1 only one user gets a full allocation while the other one gets nothing. There are thus only two options: either $(\alpha_{11}^*, \alpha_{21}^*) = (0, 1)$ or $(\alpha_{11}^*, \alpha_{21}^*) = (1, 0)$. As stated in Proposition 1, $(\alpha_{11}^*, \alpha_{21}^*)$ is continuous on $\mathcal{A}$. Together with the fact that there are only two possible options for the output which is of discrete type, this implies that F must be equal to a constant (either $(0, 1)$ or $(1, 0)$) in each connected component. Therefore, to know which value of the output corresponds to each connected component, we just need to choose one point in that connected component of $\mathcal{A}$ and solve the optimization problem for that point.

Let use consider the first connected component of $\mathcal{A}$ which is defined by $\frac{r_{12}}{r_{11}} > 1$ and $\frac{r_{12}}{r_{11}} > \frac{r_{22}}{r_{21}}$. In this region, $(\alpha_{11}^*, \alpha_{21}^*) = (0, 1)$. Indeed, let us choose one input point in this connected component such that it satisfies $r_{11} > 1$. Since either $\alpha_{11}^* = 0$ or $\alpha_{11}^* = 1$, we just need to compare $\max O_{|\alpha_{11}=1}(\alpha_{12})$ and $\max O_{|\alpha_{11}=0}(\alpha_{12})$. We have

$$O_{|\alpha_{11}=1} = \log(r_{11} + \alpha_{12} r_{12}) + \log((1 - \alpha_{12}) r_{22}) := g(\alpha_{12}). \tag{B.17}$$

This is a function of one variable α_{12}, we have:

$$g'(\alpha_{12}) = 0 \Longleftrightarrow \alpha_{12} = \frac{1 - r_{11}}{1 + r_{12}}. \tag{B.18}$$

We have $\alpha_{12} = \frac{1 - r_{11}}{1 + r_{12}} < 0$ since $r_{11} > 1$. Therefore the optimal α_{12}^* is stay in the boundary, i.e,

$$\begin{aligned} \max O_{|\alpha_{11}=1} &= \max\left(O_{|\alpha_{11}=1,\alpha_{12}=1}, O_{|\alpha_{11}=1,\alpha_{12}=0}\right) && \text{(B.19)} \\ &= \max\left(\log(r_{11}) + \log(r_{22}), -\infty\right) && \text{(B.20)} \\ &= \log(r_{11}) + \log(r_{22}). && \text{(B.21)} \end{aligned}$$

On the other hand,

$$O_{|\alpha_{11}=0} = \log(\alpha_{12} r_{12}) + \log(r_{21} + (1 - \alpha_{12}) r_{22}). \tag{B.22}$$

Combining with the condition $\frac{r_{12}}{r_{11}} > \frac{r_{22}}{r_{21}}$ we obtain

$$\begin{aligned} O_{|\alpha_{11}=0,\alpha_{12}=1} &= \log(r_{12}) + \log(r_{21}) && \text{(B.23)} \\ &> \log(r_{11}) + \log(r_{22}) && \text{(B.24)} \\ &= \max O_{|\alpha_{11}=1}. && \text{(B.25)} \end{aligned}$$

This implies $\alpha_{11}^* = 0$ in this case.
Similarly, for the connected component defined by $\frac{r_{22}}{r_{21}} > 1$ and $\frac{r_{12}}{r_{11}} < \frac{r_{22}}{r_{21}}$ we have $\alpha_{11}^* = 1$.

Conclusion:

- $\mathcal{B}$ contains the box restricted by $0 \leq \frac{r_{12}}{r_{11}} \leq 1$ and $0 \leq \frac{r_{22}}{r_{21}} \leq 1$ and the line $\frac{r_{22}}{r_{21}} = \frac{r_{12}}{r_{11}}$. $\mathcal{A}$ is the remaining region.
- As proven in Proposition 1, the allocation is continuous on $\mathcal{A}$, and moreover
  - on the region defined by $\frac{r_{12}}{r_{11}} > 1$ and $\frac{r_{12}}{r_{11}} > \frac{r_{22}}{r_{21}}$, $\alpha_{11}^* = 0$,
  - on the region defined by $\frac{r_{22}}{r_{21}} > 1$ and $\frac{r_{12}}{r_{11}} < \frac{r_{22}}{r_{21}}$, $\alpha_{11}^* = 1$.

  Combining with the formulas obtained in the three above cases for the set $\mathcal{B}$, we can see that the set of all discontinuities is the line $\frac{r_{12}}{r_{11}} = \frac{r_{22}}{r_{21}}$.
- $(\alpha_{11}^*, \alpha_{21}^*)$ is uniquely defined except on the line $\frac{r_{22}}{r_{21}} = \frac{r_{12}}{r_{11}}$.

Appendix C. Proof of Proposition 3

We first prove the first assertion. From the third condition of (KKT), for each time slot j, there exists at least one i_0 such that $\rho_{i_0 j}^* = 0$. Combining with the fourth condition of (KKT) we obtain that $\lambda_j^* = \max_i \frac{r_{ij}}{C_i^*}$, that is, the maximum is attained because of the existence of i_0. Since $\mathbf{C}^*$ is unique and continuous in R, it implies the uniqueness and continuity of λ^*.

Regarding the second assertion, it follows from the fourth condition of (KKT) that

$$\rho_{ij}^* = \lambda_j^* - \frac{r_{ij}}{C_i^*}. \tag{C.1}$$

Since $\mathbf{C}^*$ and λ^* are unique and continuous in R, it implies the uniqueness and continuity of ρ^*.