UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Self-organised Admission Control for Multi-tenant 5G Networks

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Gerard Benavides Mantecón

In partial fulfilment

of the requirements for the degree of

MASTER IN TELECOMMUNICATIONS ENGINEERING

Advisor: Oriol Sallent Roig

Barcelona, September 2017

Preface

First, I would like to express my gratitude to my supervisor, Prof. Oriol Sallent, for his guidance and constructive feedback.

My most sincere thanks to Prof. Jordi Perez for his helpful advices.

Last, but not least, I could not have succeeded without the support of my lovely family.

Gerard Benavides Mantecon

Contents

Abstract

The vision of the future 5G corresponds to a highly heterogeneous network at different levels, including multiple Radio Access Technologies (RATs), multiple cell layers, multiple spectrum bands, multiple types of devices and services, etc. Consequently, the overall RAN planning and optimization processes that constitute a key point for the success of the 5G concept will exhibit tremendous complexity.

In this direction, legacy systems such as 2G/3G/4G already started the path towards a higher degree of automation in the planning and optimization processes through the introduction of SON functionalities. SON refers to a set of features and capabilities designed to reduce or remove the need for manual activities in the lifecycle of the network. With the introduction of SON, classical manual planning, deployment, optimization and maintenance activities of the network can be replaced and/or supported by more autonomous and automated processes, operating costs can be reduced and human errors minimized.

In this work, a self-organizing admission control algorithm for multi-tenant 5G networks is proposed and developed with novel artificial intelligence techniques. A simulation-based analysis is presented to assess the improvements of the proposed approach with respect to a baseline scheme.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

2/3/4/5G	Second/Third/Fourth/Fifth Generation
3GPP	3$^{\text{rd}}$ Generation Partnership Project
ABS	Almost Blank Subframe
AC	Admission Control
ACO	Ant Colony Optimization
AI	Artificial Intelligence
ANN	Artificial Neural Network
CAPEX	Capital Expenditure
COC	Cell Outage Compensation
DoF	Degrees of Freedom
eICIC	enhanced Inter-Cell Interference Coordination
E-RAB	Evolved-Radio Access Bearer
FIS	Fuzzy Inference System
FQL	Fuzzy Q-Learning
GA	Genetic Algorithm
GWCN	Gateway Core Network
HeNB	Home evolved NodeB
HetNets	Heterogeneous Networks
HSPA	High-Speed Packet Access
KPI	Key Performance Indicator
LOS	Line-Of-Sight
LTE	Long Term Evolution
OMC	Operation and Management Center
OPEX	Operational Expenditure
MDP	Markov Decision Process
ML	Machine Learning

MNO	Mobile Network Operator
MOCN	Multi-Operator Core Network
NGMN	Next Generation Mobile Networks
NLOS	Non Line-Of-Sight
QoS	Quality of Service
Q-Value	Quality Value
RAB	Radio Access Bearer
RAN	Radio Access Network
RAT	Radio Access Technology
RB	Resource Block
RF	Radio Frequency
RL	Reinforcement Learning
SAGBR	Scenario Aggregated Guaranteed Bit Rate
SCaaS	Small Cell as a Service
SDN	Software Defined Network
SI	Swarm Intelligence
SLA	Service Level Agreement
SOCRATES	Self-Organizing Computational substRATES
SON	Self-Organizing Networks
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
WP	Work Package

Symbols

a_i	Specific action for the i-th rule
$a(t)$	Global action to be executed
$C(s)$	Nominal capacity share of the s-th tenant
d	Inter-Site Distance
f	Frequency
G	Base station antenna gain
N	Number of cells
$P(s'\|s,a)$	State transition probability
$P_{block}(T_i)$	Blocking probability of the tenant T_i
P_r	Cumulative-time system failure probability
P_t	Transmitted power per RB
$q[i,j]$	Associated q-value to the rule i and action j
$Q(s,a)$	Q-value for the state s and action a
$r(t)$	Total reinforcement signal
$\hat{r}(n)$	Estimated bit rate per RB
r^{t+k}	Instantaneous reward at the $t+k$ iteration
$r_s(t)$	Reinforcement signal contribution of the s-th tenant
R^k	Cumulative reward at the k-th time instant
R_{req}	Required bit rate
S	Number of tenants
$V_t(s(t+1))$	Value of the new transitioned state
$\alpha_i(s(t))$	Activation function for the i-th rule
γ	Discount factor
$\Delta C(s,n)$	Variable that considers the possible unused capacity left by other tenants
$\Delta C_b(s,n)$	Variable that ensures capacity share balance across all the cells
$\Delta C_e(s,n)$	Extra capacity which is available for the s-th tenant in the n-th cell
ΔQ	Error signal between consecutive Q-functions
$\Delta\rho$	Estimated number of RBs required by the newly admitted RAB
ϵ	Greed factor
ϵ^-	Reducing rate of ϵ
η	Learning rate
λ	Session arrival rate
μ_A	Degree of truth of a fuzzy set A
$\mu_{ij}(x_j(t))$	Membership function value for the i-th rule and j-th input
$\pi(s)$	Optimal Q-Learning policy for a given state s
$\rho(n)$	Number of available RBs in the n-th cell
$\rho_G(s',n)$	Average number of RBs assigned to the RABs of the s-th tenant
$\rho(n)\alpha_{th}(n)$	Cell-level AC threshold
σ	Shadowing standard deviation

Chapter 1

Introduction

1.1 Introduction and scope

The upcoming generation of mobile communication systems, formerly known as 5th Generation (5G), is expected to be available in 2020 with around 8 billion of worldwide mobile subscriptions [1]. The irruption of the 1st Generation (1G) of wireless cellular technology changed the world by connecting *people to people*, instead of connecting *places to places*. Now, due to the overwhelming increase of smart devices and the advent of the Internet of Things (IoT), 5G will aim to connect *everything to everything*.

Unlike its predecessors, 5G will be conceived as a set of technologies that are both efficient and economical in terms of Key Performance Indicators (KPIs). Specifically, from an operator's perspective, the following capital KPIs will be considered: capacity, quality of service (QoS), capital expenditure (CAPEX) and operational expenditure (OPEX). On the other hand, from an end user's point of view, the associated KPIs mainly include: seamless connectivity, perception of almost infinite capacity (i.e. negligible latency) and the cost of service.

In order to cope with the unavoidable demand increase of network capacity, three solution approaches are envisaged. As illustrated in Figure 1.1, the projected capacity growth in 5G, with respect its predecessor 4G, comes by either improving the spectral efficiency of wireless technologies (3x-5x), by allowing more spectrum allocation (5x-10x), by deploying more network nodes (40x-50x), or by harnessing together the three aforementioned approaches (a total of 600x-2500x capacity increase). It is also observed that the operational complexity of future 5G networks will scale linearly only with the network densification, since the other two capacity growth dimensions are expected to mostly affect the complexity of hardware design. Considering that 4G networks have typically 1500 parameters to be configured and optimized (later on defined as degrees of freedom), and 5G networks are expected to have around 2000, simple calculations lead to a (2000/1500 x (40x-50x)) $\approx$ 53x-67x increase in operational complexity [2].

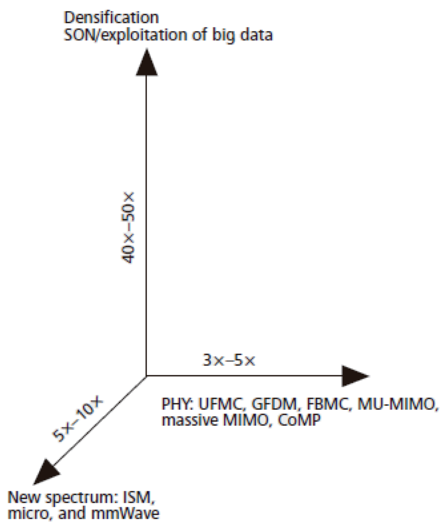
Densification
SON/exploitation of big data
40×–50×
3×–5×
PHY: UFMC, GFDM, FBMC, MU-MIMO, massive MIMO, CoMP
5×–10×
New spectrum: ISM, micro, and mmWave

Figure 1.1: Dimensions of projected capacity growth in 5G [2]

In the emerging wireless landscape, 5G networks will be required to be fully self-organizing with end-to-end network behaviour intelligence to ensure a profitable business model. In this context, the introduction of a self-organizing network (SON) engine will enable the exploitation of artificial intelligence mechanisms for the efficient management of network resources, that will allow users to perceive seamless and limitless connectivity. Hence, SONs clearly aim to reduce OPEX, by replacing the classic manual configuration, post-deployment optimization and maintenance in cellular networks with self-configuration, self-optimization and self-healing features.

Current SONs for 4G, 3G and even 2G networks usually follow the methodology illustrated in Figure 1.2, in which spatio-temporal knowledge (obtained through e.g. OAM reports, drive tests, etc.) is fully or at least partially available. For example, the location of potential coverage holes or handover ping-pong zones are assumed to be known by the SON engine. Nevertheless, the current SON approach should not be considered in future 5G networks, as it does not provide dynamic models to predict system behaviour in a live-operation fashion in order to meet the stringent latency requirements of the upcoming mobile generation [2].

Another important matter regarding 5G networks operation will be shifting from reactive to proactive SON. Current SON functions usually have a reactive behaviour, meaning that they are triggered when a problem has occurred. The actions of *observing* the environment, *diagnosing* the problem and *executing* the compensating action involve to spend a valuable operation time which is not compatible with the 5G targeted latency requirements. Therefore, the SON paradigm has to be shifted from reactive to proactive. This transformation leads to the premise of predicting the problems beforehand, by inferring network-level intelligence to take preemptive actions to resolve the issue before it occurs.

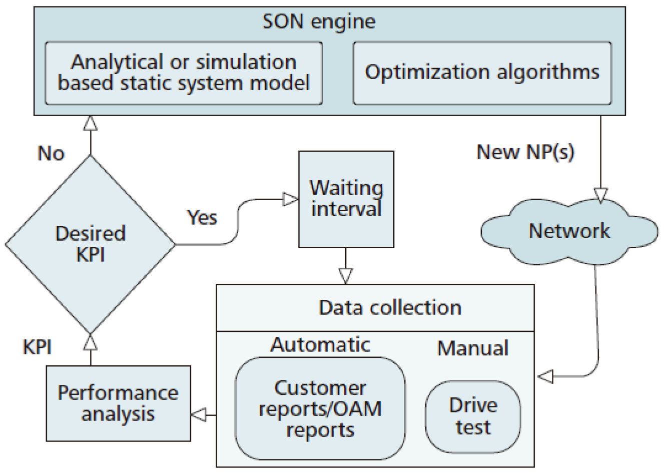

FIGURE 1.2: SON in 2G, 3G and 4G networks [2]

A possible framework for 5G SON is depicted in Figure 1.3. It is observed that big data, which can be briefly defined as the huge amount of information available from the different sources of the mobile network, is the main feature that make future SONs distinct from legacy cellular systems. The sources of big data for 5G SONs can be split in three main network-level layers: *subscriber-level data* (e.g. call success rate, call drop ratio, speech quality, IP traffic flow), *cell-level data* (e.g. received interference power, thermal noise power, channel baseband power) and *core-network-level data* (e.g. historical alarms logs, device configuration records, authentication). Besides data collection, the introduction of machine learning and data analytics tools allows the automatic transformation from big (raw) data to right (meaningful) data. Once the right data is available, system and user behaviour models can be extracted and be delivered to the SON engine to perform the appropriate SON functions.

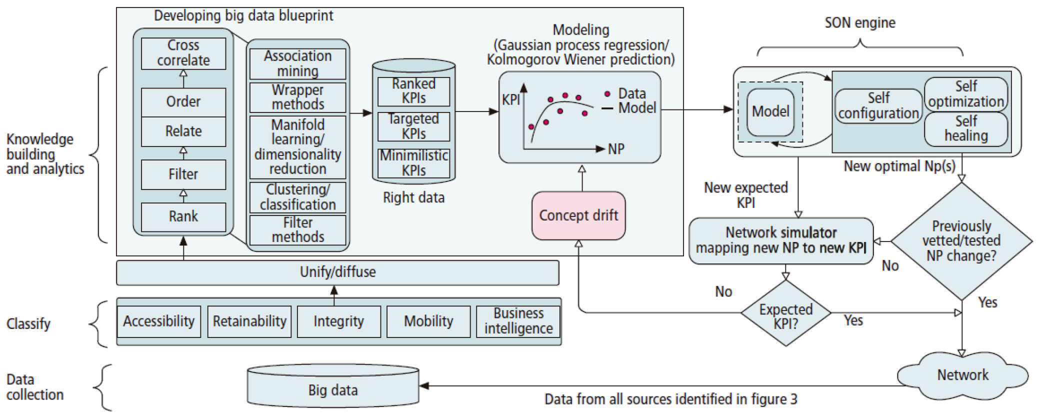

FIGURE 1.3: Expected framework for future 5G SON [2]

As aforementioned earlier, SON will enable the exploitation of artificial intelligence-based techniques (e.g. machine learning, bio-inspired algorithms, fuzzy neural networks) in order to efficiently handle the problems of large-scale complex systems.

In this work, an artificial intelligence technique is chosen to develop a self-organizing admission control algorithm for multi-tenant 5G networks. It is important to remark that the chosen approach could not be the optimal one amongst the all possible candidate solutions, but has been selected according to its innate properties to fit well in most of the self-optimization processes. In other words, the selected approach is most likely to perform well under any self-optimization problem (i.e. generalist approach).

This document is structured as follows: Chapter 2 introduces the main concepts regarding SON. Chapter 3 presents a brief survey of the current Artificial Inteligence (AI) techniques. Chapter 4 focuses on the theory of Admission Control (AC) for multi-tenant Radio Access Networks (RANs) and introduces the proposed self-optimised AC strategy for adjusting the share of resources used by each tenant. Chapter 5 presents the AI-based algorithms which have been chosen to self-optimize the AC parameters. Finally, the conclusions of the overall work can be found in Chapter 6.

1.2 Contributions and goals

Once the scope of the project is clear, a set of objectives should be defined. As aforementioned earlier, the scope is broad enough to undertake the present work in quite different ways (e.g. by considering various artificial intelligence-based techniques).

The following goals were initially settled down and successfully acomplished by the end of this project:

- Review the concept of SON and study the AI-based techniques for self-optimization on Heterogeneous Networks (HetNets).
- Understand the theory behind the Admission Control for multi-tenant Radio Access Networks.
- From the two previous goals, decide which is the most optimal self-optimization algorithm for the proposed research topic and implement it.
- Analyse the results of the AI algorithm and study its feasibility in an hypothetical SON deployment.

1.3 Work plan

The work load of this thesis has been split in three work packages (WP), which are represented in Figure 1.4 and briefly detailed below:

- **State of the art**: review of the current state of the technologies involved in the scope of this project.
- **Algorithm implementation**: formulation and development of both supervised and unsupervised learning algorithms. The analysis of the results given by these methods are also included in this WP.
- **Project management**: tasks which are related with the writing and defense of this thesis.

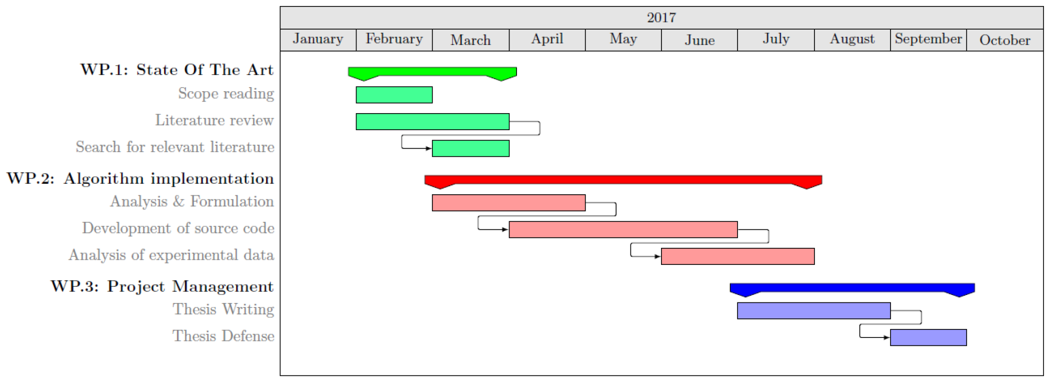

FIGURE 1.4: Gantt chart

Each WP has its internal tasks, which are shown in Table 1.1, Table 1.2 and Table 1.3, respectively.

Internal task	Start date	End date	Main objective
Scope reading	1/2/17	15/2/17	Set realistic project objectives
Literature review	8/2/17	7/3/17	Review recommended literature
Search relevant literature	8/3/17	31/3/17	Read other useful documents

TABLE 1.1: WP.1: State Of The Art

Internal task	Start date	End date	Main objective
Formulation	15/3/17	15/4/17	Formulate the algorithm's equations
Development	16/4/17	7/6/17	Implement and test the source code
Analysis	8/6/17	8/7/17	Extract conclusions of the results

TABLE 1.2: WP.2: Algorithm implementation

Internal task	Start date	End date	Main objective
Thesis Writing	15/7/17	15/8/17	Final document writing
Thesis Defense	x/9/17	-	Thesis presentation

Table 1.3: WP.3: Project Management

Besides the 30 ECTS work load, the autor of this thesis was enrolled in an elective course and was working in a part-time engineer job. Hence, the duration of the present project was extended by two additional months, with the authorization of the thesis supervisor.

Chapter 2

Self-Organizing Networks

2.1 Definition

Self-Organizing Networks are defined as a set of use cases covering all aspects of network operation, from network planning to maintenance activities. SONs will lead network intelligence and network management features in order to seamlessly automate the configuration and optimization of wireless networks [3].

In order to justify the need of implementing SONs in the future 5G networks, the term degrees of freedom (DoF) will be shortly introduced. System's DoF are defined as the amount of parameters which can be tweak in any wireless network. It's important to mention that some DoF could be highly dependent among themselves.

The implementation of newer radio access technologies (RATs) exponentially increases the number of DoF as follows (logarithmic scale) [4]:

- New system DoF $\rightarrow$ x20-30 old system DoF
- New system density $\rightarrow$ x4 old system density
- New DoF/km^2 $\rightarrow$ x100 old DoF/km^2 (x2 orders of magnitude)

Some basic examples of DoF are: transmit power, carrier frequency, to which cell a handover should be performed, time delay until a handover is executed, etc.

Nevertheless, it is not just an increase of DoF that can be optimized in a single RAT, but also in all the available technologies that are operating at the same time (e.g. GSM/UMTS/LTE/LTE-A). At this point, the concept of Heterogeneous Networks (HetNets) arises.

HetNets are defined as a wireless networks holding a vast variety of RATs, formats of cells and many other aspects, aiming to combine them to operate in a seamless way (Figure 2.1). Hence, the DoF increase significantly with the emerging HetNets and thus, the probability that things go wrong (e.g. coverage outages, handover failures).

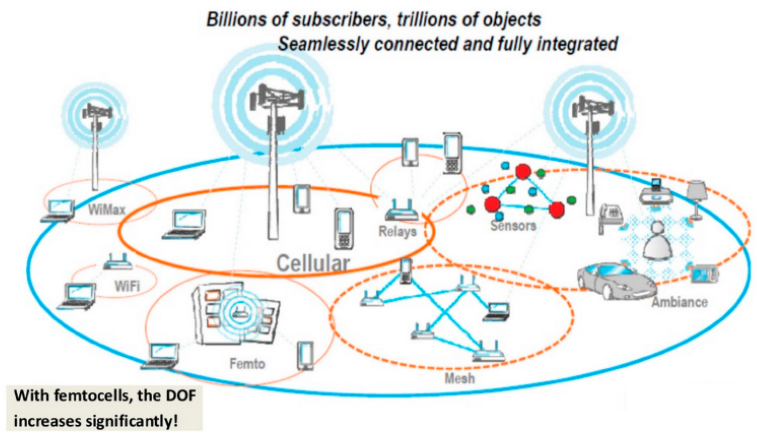
Billions of subscribers, trillions of objects
Seamlessly connected and fully integrated
WiMax
Cellular
Relays
Sensors
Ambiance
WiFi
Femto
Mesh
With femtocells, the DOF increases significantly!

Figure 2.1: Illustration of a Heterogeneous Network (HetNet) [4]

The probability that any system fails is directly related to the DoF. For instance, LTE has around 1500 DoF [2], meaning that its cumulative-time system failure probability P_r would be close to 1 since the network deployment (Figure 2.2), considering that its DoF are highly dependent among themselves (i.e. series system). In other words, the system is likely to fail from the very beginning if nothing is done in order to prevent it.

SONs aim to mitigate the consequences of DoF within HetNets and improve the scalability of the whole system, by reducing the lifecycle cost (O/CAPEX) and by dynamically optimizing the radio network performance during operation (e.g. improved user experience). Figure 2.3 illustrates the impact of employing SON functionalities in the different stages of a typical wireless operator's workflow.

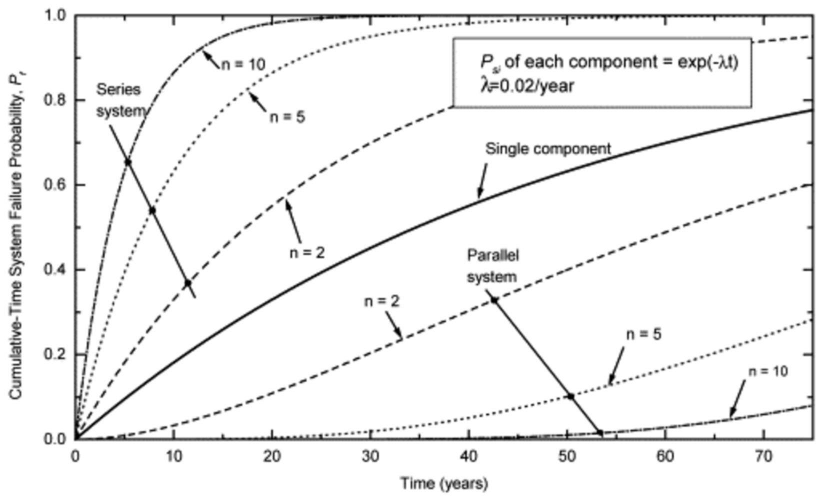

Figure 2.2: Cumulative system failure probability as a function of time [5]

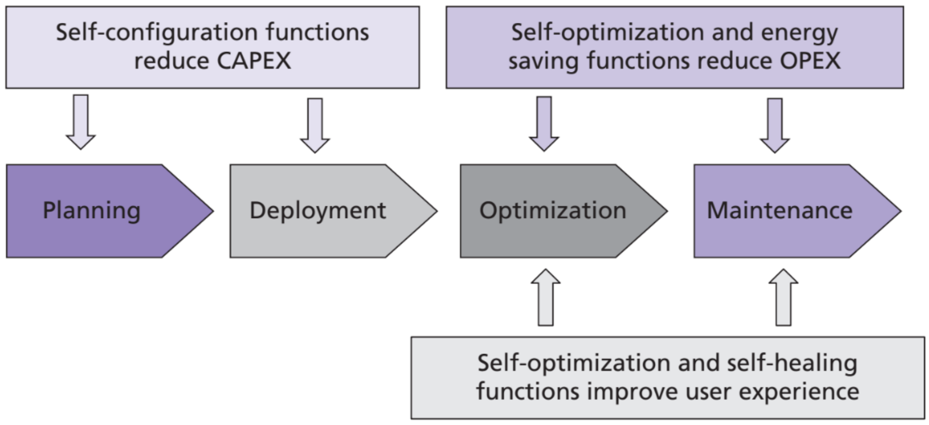
Self-configuration functions reduce CAPEX
Self-optimization and energy saving functions reduce OPEX
Planning
Deployment
Optimization
Maintenance
Self-optimization and self-healing functions improve user experience

FIGURE 2.3: Impact of SON functions on a typical wireless operator's workflow [6]

Besides the above described answers to the main questions "*what* are SONs?" and "*why* are they needed?", it is worth to mention three important elements of SONs:

- Elements of **autonomous**. As the number of base stations (e.g. home-evolved NodeBs or femtocells) will be at least quadruplicated by the time 5G is implemented, there rises a need to configure and manage them with the least possible human interaction, hence further reducing both capital and operational expenditure costs, from a network operator point of view.

- Elements of **intelligence/cognition**. *A system is intelligent, if it is able to operate under conditions it was not originally designed for* [4]. A threshold based system is not well suited for the upcoming 5G networks since there is no intelligence in that, it is just deterministic. Hence, SONs are required to adapt, learn and build intelligence on prior observation.

- Elements of **optimality**. Despite large DoF, SON has to be (close to) optimal. For instance, SON could have an outstanding RF physical layer technology which underperform due to poor resource management, thus moving its performance to an undesired (sub-optimal) region.

Although the concept of SON was formerly defined in 2008 (3GPP Release 8), SON techniques had been previously discussed and even tried in 2G systems in the early 2000's. Nowadays, many HSPA infrastructure vendors offer a set of automatic optimization algorithms. However, SON has not been standardized in the legacy systems and all available techniques are fully vendor dependent. Hence, 3GPP will address the standardization of self-organizing features of the future 5G networks.

2.2 Architectures

From the previous section (2.1 Definition), an additional question regarding SON operation may arise: *where* is self-organization carried out? Or in other words: *where* are the decisions taken?

There are four identified levels of SON execution (i.e. architectures), which are shown in Figure 2.4 and briefly described next [4]:

1. **Localised**. Autonomous SON execution based on purely local information at (Home)eNodeB[1] and the User Equipments (UEs) associated with that (H)eNB. An example of a SON feature running through a localised architecture could be the following one: an UE reports to its eNB through the link quality indicator that it has a very weak field strength. The associated eNB would take the local decision of downgrading the modulation index, meaning that there is no need in involving a central entity placed hundred of kilometres away.

2. **Distributed**. Autonomous SON execution based on information exchanged with neighbouring (H)eNB (e.g. via X2 interface). A particular example to justify the use of a distributed architecture would be the SON feature of load balancing: if a distributed eNB is heavily loaded, it can request to its neighbours their situation in terms of traffic load. If the request is favourable, the affected eNB could handover some users to the requested cells.

3. **Centralized**. Decision taking based on fairly complete system information. An optimal SON algorithms are required in order to manage large amounts of collected information and extract meaningful conclusions about the state of the network.

4. **Hybrid**. Any mixture of the above mentioned levels of SON execution. It is usually the best fitted approach for many applications, as it handles well the trade-off amongst the aforementioned architectures.

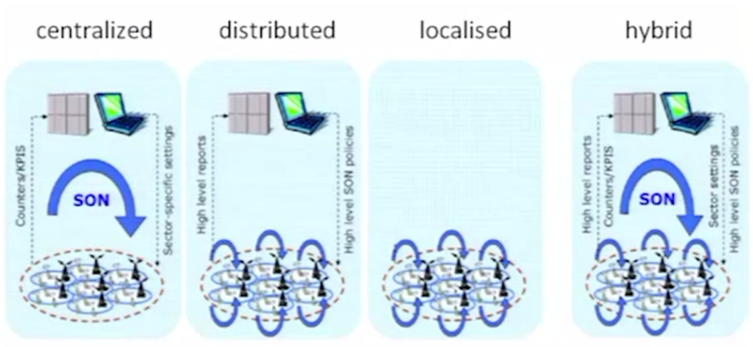

Figure 2.4: The four levels of SON execution [4]

[1] 3GPP's term for an LTE femtocell or Small Cell

Each architecture approach has its own advantages and drawbacks. Figure 2.5 summarizes the main characteristics of the aforementioned levels of SON execution and the trade-off between them. It is observed that SON features running towards localised architectures are much quicker in terms of execution, since there is no need to feed the information to central entities. Furthermore, both distributed and localised approaches are less prone to the single point of failure. On the other hand, moving towards centralized architectures, Mobile Network Operators (MNOs) are able to collect a vast amount of information about the state of their networks, meaning that they are able to take better decisions.

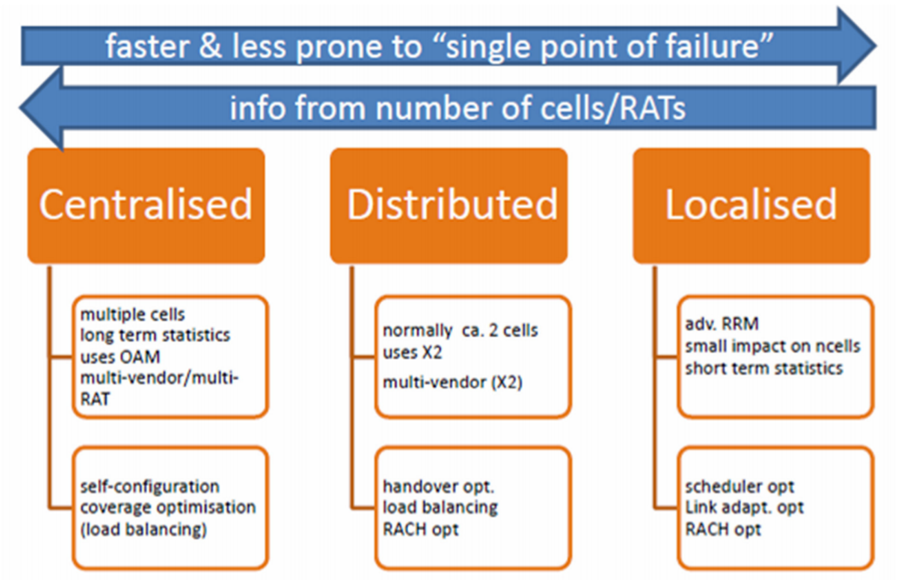

FIGURE 2.5: Trade-off between the proposed SON architectures [4]

2.3 Self-X Functions

To end up this chapter, a small introduction about the current Self-X functions, also known as 3GPP SON features, is shortly presented.

The already well known purpose of SONs is to seamlessly integrate network planning, configuration and optimization into a single automated process, requiring minimal manual intervention in the lifecycle of the network. Hence, the introduction of SON features (Figure 2.6) will allow mobile operators to reduce their operating costs while minimizing human errors. More specifically, there are three main Self-X functions proposed by the 3GPP [7]:

- **Self-Configuration**. Automation of the processes involving newly deployed cells, such as their configuration and authentication, besides the adjustment of their parameters (e.g. transmission power, inter-cell interference), in a plug and play fashion. Moreover, preceding the aforementioned procedure, a Self-Planning stage (e.g. ascertaining cell locations) is usually considered within the Self-Configuration feature as well.

- **Self-Optimization**. Ability of the network to keep improving its performance in terms of coverage, capacity and service quality, by iteratively optimizing the different network settings, taking into account the radio characteristics, traffic dynamics and user demands, among other aspects. Many Self-Optimization use cases are envisaged for future mobile networks, such as the ones proposed by the 3GPP, which are mainly focused on aspects concerning load balancing, power adaptation, neighbourhood maintenance and mobility management, or the ones discussed in SOCRATES/NGMN projects, in which QoS optimization (e.g. admission control, congestion control) is the principal research topic.

- **Self-Healing**. Set of processes designed to allow seamless maintenance and enable network to recover from failures in an autonomous fashion. In fact, autonomous re-configuration is considered as an important feature of Self-Healing in HetNets. By studying the behaviour of users and observing the changes of network conditions, HetNets will require from real-time re-configurations without termination of mobile services.

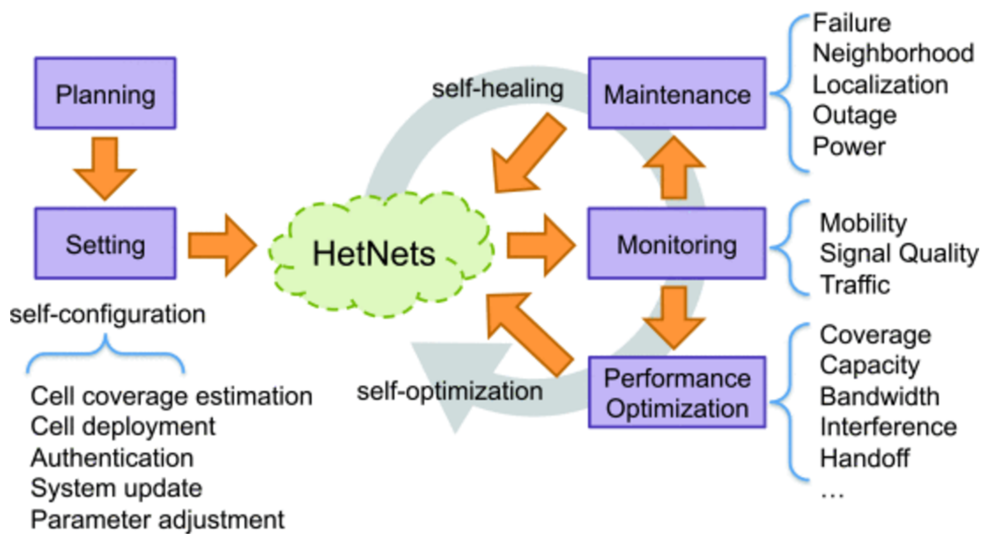

Figure 2.6: Self-X Functions for HetNets [7]

Chapter 3

Artificial Intelligence-Based Techniques for HetNets

"*Brains exist because the distribution of resources necessary for survival and the hazards that threaten survival vary in space and time*" (J. Allman, 2000, p.2). This idea can be related with the addition of intelligence in the aforementioned large-scale complex systems, whose environment progressively demands efficient strategies (e.g. in terms of optimization) in order to meet the expected requirements.

As discussed earlier, HetNets are becoming quite challenging to deal with, since the number of network resources keeps steadily increasing. AI techniques are aimed to overcome the drawbacks of large-scale systems and, therefore, their implementation would add intelligence to the current and future HetNets to reduce human involvement, which is one of the main goals of SON. Then, AI-based techniques can substantially reduce the operational and capital expenditure (O/CAPEX) and optimize network capacity, coverage and Quality of Service (QoS) in HetNets, according to the Self-X features [7].

AI techniques share the main objective of turning emerging HetNets smarter, but they can be quite different among themselves, by the means of operability. Some are inspired from nature findings (e.g. Bio-inspired Algorithms), a few of them are motivated by the ways of human reasoning (e.g. Fuzzy Systems) and some others are based on local interactions and recursive feedback-based learning (e.g. Machine Learning). A careful study of each technique and its feasibility in certain network applications should be carried out in order to understand each one's advantages and drawbacks.

In the present chapter, the state of the art of the most relevant AI-based techniques that are being studied for their deployment in emerging HetNets will be presented, with special emphasis towards the methods that have been chosen to carry out the self-organised AC for multi-tenant 5G networks algorithm (section 3.1 and section 3.3).

3.1 Machine Learning

The term Machine Learning (ML) is relatively new. It was defined back in 1959 by A. Samuel, a pioneer of AI research, as *"the ability to learn without being explicitly programmed"*. Thus, ML is envisaged for specific range of applications where designing explicit algorithms with the expected performance is not even feasible. Learning methods are needed whenever knowledge (e.g. human expertise) does not exist, or it is somewhat hard to acquire. Nevertheless, ML algorithms are able to exploit training data or past experience in order to build useful models (e.g. patterns or policies) for a wide range of applications.

Nowadays, there are many successful ML-based applications in various disciplines. For instance, retail companies collect past sales data to analyze customers preferences, thus improving their service. Financial institutions consider past transactions to predict customers credit risks. Most of email applications, regardless of their popularity, use ML to decide whether an incoming message should be considered as spam or not. In bioinformatics, the huge amount of data available can only be analyzed and its knowledge extracted using what is known as data mining [1] [8].

Among many ML techniques, Reinforcement Learning (RL) is inspired by behaviorist psychology [9], where an agent tries to learn from its environment the best set of actions to maximize the desired system performance (i.e. cumulative rewards, see Equation 3.1).

$$R^k = r^k + \gamma r^{k+1} + \gamma^2 r^{k+2} + \gamma^3 r^{k+3} ... = \sum_{t=0}^{\infty} \gamma^t r^{t+k} \tag{3.1}$$

where R^k is the cumulative reward at the k-th time instant (i.e. iteration). r^{t+k} denotes the instantaneous reward obtained as a consequence of executing an action at the $t+k$ iteration. γ is the discount factor, where values close to 0 mean that the agent tries to optimize immediate rewards (i.e. the agent is *myopic*), whereas values near 1 consider long-term high rewards.

Shortly, RL effectively learns the system impact $\mathbb{I}$ of a particular action $\mathbb{A}$, with the objective of maintaining a determined performance metric $\mathbb{P}$, based on a particular experience $\mathbb{E}$, where the system aims to iteratively improve its performance $\mathbb{P}$ while executing action $\mathbb{A}$, again by exploiting its experience $\mathbb{E}$. The model built may be predictive to make future predictions, or descriptive to gain knowledge from data, or both of them [10].

[1] Data mining is the computing process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems

In Figure 3.1, it is illustrated a possible RL-based optimization scheme whose agent carries out the aforementioned paragraph learning procedure, where the instantaneous reward could be any typical Key Performance Indicator (KPI), such as call blocking ratio or cell-edge coverage, depending on the target application.

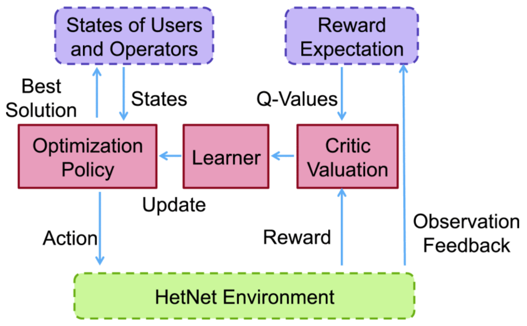

FIGURE 3.1: Learning-based optimization scheme in HetNets [7]

An RL technique which is increasingly receiving substantial attention both in the academic as well as industrial communities is Q-Learning. Its aim is to find an optimal Quality Value (Q-Value) for any given Markov Decision Process (MDP) by experiencing the consequences of actions [11].

MDPs provide a mathematical framework for modeling decision making in specific situations, where the outcomes are partly random and partly under the control of a decision maker, as illustrated in Figure 3.2 (a). The probability that the process moves into its new state s' is influenced both by the specific action chosen, as well as by the system inherent transitions, formally described by the state transition probability $P(s'|s,a)$. By contrast, Q-Learning lacks of system transition model knowledge, as shown in Figure 3.2 (b). Nevertheless, the reason of using this RL technique is that it is able to compare the expected utility of the available actions without actually requiring a model of the environment.

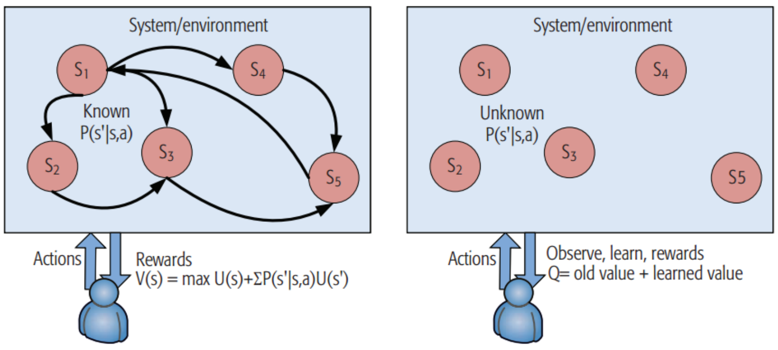

FIGURE 3.2: (a) Markov Decision Process (MDP) and (b) Q-Learning [10]

Q-Learning is based on the straightforward premise that the optimal policy is the one which selects the set of actions with the highest long-term reward. This can be mathematically expressed as follows:

$$\pi(s) = argmax_a Q(s,a) \tag{3.2}$$

where the policy π, for a given state s, selects the action a that maximizes the Q-value $Q(s,a)$.

The core algorithm of Q-learning iteratively updates the Q-value for each state based on the experience of the actions, namely rewards. The equation below describes the algorithm for one-step Q-Learning [12]:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \eta[r^{t+1} + \gamma max_a Q(s_{t+1},a) - Q(s_t,a_t)] \tag{3.3}$$

where $Q(s_t,a_t)$ is the Q-function to be updated, based on the state s and action a in iteration t. The learning rate $0 \leq \eta \leq 1$ determines the degree of membership of newly acquired information (e.g. $\eta \simeq 1$ would only consider the most recent information). r^{t+1} refers to the numerical reward received by the agent after transitioning from the state s_t to s_{t+1}. As commented earlier in Equation 3.1, γ is the discount factor ($0 \leq \gamma \leq 1$), thus the same concept applies. Finally, $max_a Q(s_{t+1},a)$ is the value of the action a that is estimated to return the largest total future reward, based on all possible actions for the next state s_{t+1}.

Briefly, Equation 3.3 simply updates the existing Q-value by adding the difference between the old estimate of future reward and the new estimate, multiplied by η.

In some optimization applications, however, continuous states and action spaces may lead to extremely complex scenarios. To mitigate this problem, the incorporation of fuzzy logic (commented in Section 3.3) in RL techniques (e.g. Fuzzy Q-Learning) has been widely used to discretize the state and action spaces. Therefore, instead of having an undefined number of states and actions, fuzzy logic limits these numbers accordingly from what it is required in each application without compromising neither the convergence time nor the accuracy of the states.

One of the main drawbacks of Q-Learning is its innate time (usually long) needed to achieve the best policy. The agent takes actions throughout each optimization iteration with the goal of improving the accuracy of the Q-Values. Initially, a default policy should be defined (e.g. choose random actions). Then, the agent follows this policy until it converges towards the optimal action-value Q^*. Depending on the complexity of the optimization scenario, higher convergence times could not reach the Mobile Network Operator (MNO) expectations.

RL techniques have been already studied and its utility been proven in all the discussed Self-X functionalities. A few examples found in the literature are shortly described below:

- **Self-Configuration**: Due to the increasingly amount of devices in HetNets, interference management strategies can benefit from RL techniques, as discussed in [13], where each cell optimizes its Almost Blank Subframe (ABS), instead of relying on predefined configuration, to achieve time-domain adaptive enhanced inter-cell interference coordination (eICIC). Similar distributed interference optimization approach is studied in [14].

- **Self-Healing**: The study presented in [15] proposes an adaptive policy-based framework. In this case, reinforcement learning helps to create and update policies dynamically in response to changing reconfiguration requirements by observing changes of users and network conditions.

- **Self-Optimization**: Many optimization tasks are envisaged for the implementation of smarter HetNets. For instance, the study in [16] proposes self-optimization of antenna tilt and power through cooperative reinforcement learning (e.g. to avoid local selfish rewards), whose objective is to jointly optimize network coverage and capacity (CCO). An hybrid architecture is adopted (Figure 3.3 (a)), where each entity runs in a distributed manner to achieve fast adaptation while a central entity takes over the cooperation of distributed optimization. The simulation results show that RL techniques perform significantly better (Figure 3.3 (c)) than the best fixed configuration available (Figure 3.3 (b)).

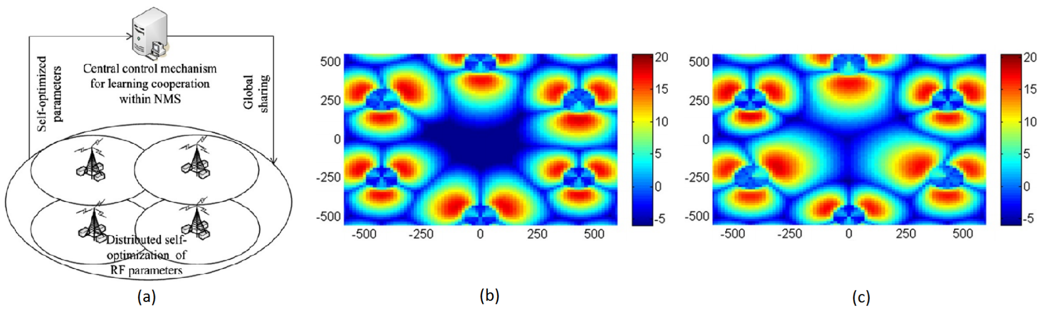

FIGURE 3.3: (a) Hybrid SON architecture for CCO, Simulated network SINR distribution under abrupt changes (b) To be optimized and (c) Optimized [16]

3.2 Bio-inspired Algorithms

Inspiration from nature findings has lead to successful algorithmic approaches to face optimization problems amongst different research disciplines, such as computational biology or telecommunications. The efficiency of bio-inspired algorithms is strongly tied with the effectiveness of the best features in nature, especially the selection of the fittest in biological systems that have evolved by natural selection [17].

Metaheuristic [2] algorithms are frequently nature-inspired, and their applicability in Self-X functions has been (and will be) studied. The most relevant metaheuristics for the emerging HetNets include Evolutionary Algorithms (e.g. Genetic Algorithm), Swarm Intelligence (e.g. Ant Colony Optimization) and Artificial Neural Networks (ANN).

The description of the above mentioned AI-based techniques are described in subsection 3.2.1 (Genetic Algorithms), subsection 3.2.2 (Ant Colony Optimization) and subsection 3.2.3 (Artificial Neural Networks), respectively.

3.2.1 Genetic Algorithms

Genetic Algorithms (GAs) are well suited for multi-objective optimization problems. Hence, HetNets might benefit from GAs in cell planning or node placement optimization problems, where a large set of parameters need to be evaluated. It has been proven that GAs are quite efficient in solving problems whose complexity is high, and the time needed to converge to optimal (or suboptimal) results is usually low, compared with other bio-inspired algorithms [18].

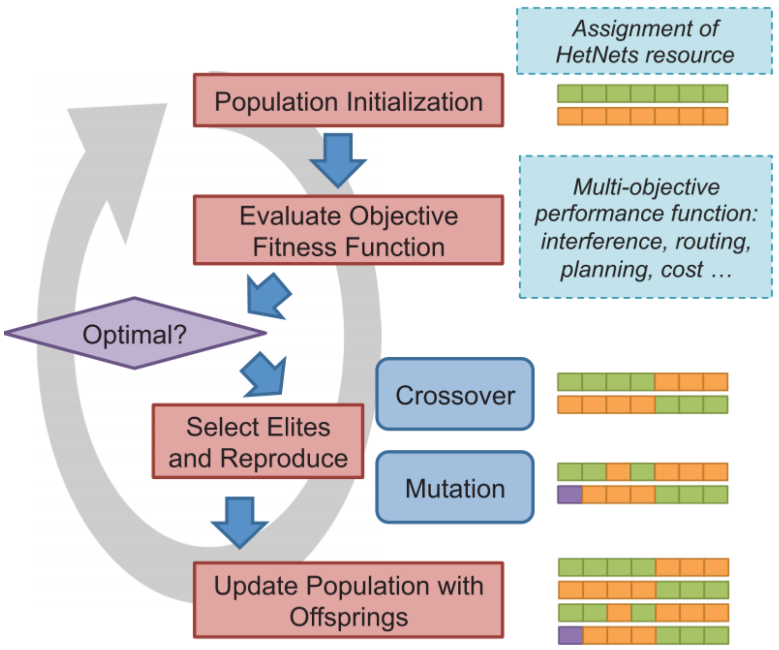
Assignment of HetNets resource
Population Initialization
Evaluate Objective Fitness Function
Multi-objective performance function: interference, routing, planning, cost …
Optimal?
Crossover
Select Elites and Reproduce
Mutation
Update Population with Offsprings

Figure 3.4: Genetic Algorithm (GA) optimization flow for HetNets [7]

[2]A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms (Sörensen and Glover, 2013).

Figure 3.4 briefly illustrates the main optimization procedure carried out by GAs in HetNets scenarios [19]:

1. GA starts by creating an initial population of chromosomes (e.g. HetNet resources), which are the basic building blocks of the algorithm.

2. Each chromosome encodes a solution of the problem, and its fitness value is linked to the value of the multi-objective performance function (e.g. interference, routing, planning, cost, ...).

   A chromosome consists of genes that can be represented in the form of a binary or integer string. For example, as illustrated in Figure 3.5, the first three bits represent the network ID (010) and last three bits are the channel ID (110).

   The fitness measure (i.e. survival measure) evaluates each individual chromosome by determining how well they solve the given problem. The fitness is normally represented by a real number, where higher values mean that the chromosome is closer to the optimal solution.

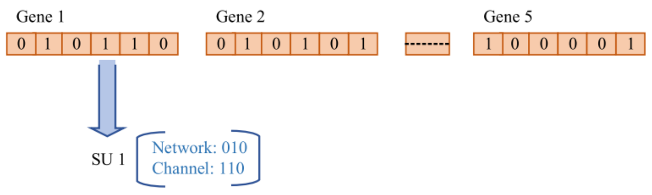

FIGURE 3.5: GAs chromosome mapping for network and channel selection [19]

3. GA adopts two gene evolution tasks to potentially find better solutions:

   a) *Crossover*, also known as reproduction, aims to combine two random parent chromosomes, where their characteristics are exchanged with each other to form a pair of child chromosomes. For example, as shown in Figure 3.6, two parent chromosomes p_1 and p_2 crossover and produce two child chromosomes c_1 and c_2.

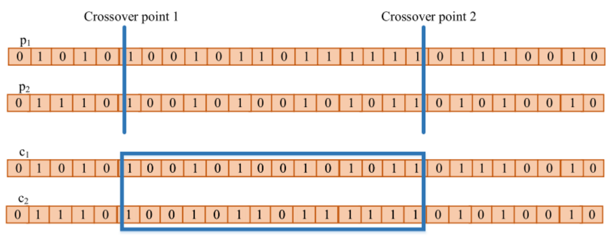

FIGURE 3.6: 2-point crossover procedure for generating child chromosomes [19]

   b) *Mutation* reorganizes the structure of genes in a chromosome randomly so that a new combination of genes may appear in the next generation. It is applied to the child chromosomes, altering a binary bit of 0 to 1 or vice versa. This action prevents falling to regional optimal solutions.

4. The offspring inherits the gene of superior chromosomes and eliminates poor chromosomes through competition. Thus, the quality of the population may be improved after each generation.

From the aforementioned optimization procedure, a more intuitive and simplified GA pseudocode arises (Algorithm 1).

```
begin
    $t \leftarrow 0$;
    initialize population $P(t)$;
    evaluate $P(t)$;
    while not termination-condition do
        begin
            $t \leftarrow t+1$;
            select $P(t)$ from $P(t-1)$;
            alter $P(t)$;
            evaluate $P(t)$;
        end
    end
end
```

Algorithm 1: GA pseudocode [20]

Evolutionary algorithms can be found in the literature to solve mainly cell planning and node placement optimization problems. For example, the study in [21] implements a multi-objective GA to address a communication nodes placement problem in HetNets. Its goal is to maximize the communication coverage as well as the total capacity bandwidth, while minimizing the placement cost. Similarly, the studies in [22] and [23] propose an evolutionary multi-objective algorithm for 4G base station planning, where signal coverage, system capacity and cost are taken as objective functions and interference is considered as a very relevant constraint. Finally, the work in [24] uses a distributed GA to to dynamically optimize the coverage of a femtocell group by adaptively adjusting the pilot power.

3.2.2 Ant Colony Optimization

Ant Colony Optimization (ACO) is one of the most known algorithms of Swarm Intelligence (SI). The term swarm is commonly used for the assembling of animals (e.g. insect colonies) who perform collective activities. Each individual of a swarm act without supervision, and they behave stochastically due to their perception within the neighbourhood. Hence, self-organization is the key characteristic of a swarm system [20].

The main principles to be satisfied by a swarm algorithm to have an intelligent behaviour are [25]:

- **The proximity principle**. The swarm should be able to do simple space and time computations.
- **The quality principle**. The swarm should be able to respond to quality factors in the environment, such as the quality of food or safety of location.
- **The principle of diverse response**. The swarm should not allocate all of its resources along excessively narrow channels and it should distribute resources into many nodes.
- **The principle of stability**. The swarm should not change its mode of behaviour upon every fluctuation of the environment.
- **The principle of adaptability**. The swarm must be able to change behaviour mode when it matters.

The ACO is inspired by the behaviour of ants and their ability to find the shortest path towards an objective (e.g. food). In any ant colony, every ant collaborates in discovering optimal routes based in an accumulative pheromone trail.

Figure 3.7 shows an illustrative example of the ant colony behaviour:

(a) Ants choose uniformly left/right-side paths in their search for food source.

(b) Ants that have chosen the shortest path (i.e. H-B-F) will return earlier back to home, whereas the other half pack of ants will be still on their way to the food source.

(c) Considering that all the ants move at a constant speed, those which have chosen the aforementioned shortest path will deposit more pheromone and thus it will be steadily accumulated along the optimal route.

(d) Ants notice that the shortest path is the one with more pheromone accumulated. Therefore, the longest H-A-F route is progressively discarded.

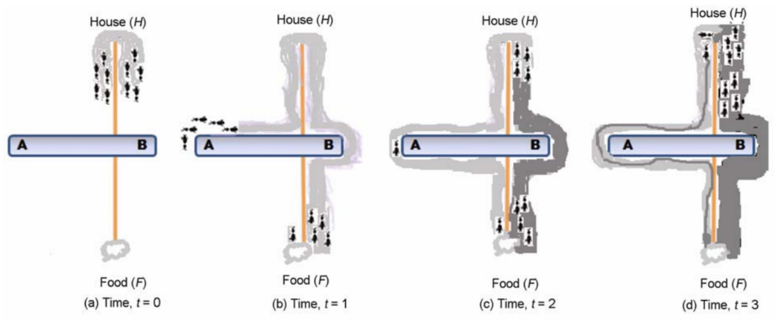

FIGURE 3.7: Ant colony principle [20]

The existing similarities between the ant colony and networking systems, such as an ant corresponding to a data packet and an ant trail corresponding to a networking route, have lead ACO to be proposed as a novel methodology to achieve optimal performance in interference management, routing and coverage problems [26].

Figure 3.8 illustrates a possible ACO procedure for HetNets:

1. The system is randomly initialized with a population of individuals, where each individual represents a particular decision point (e.g. signaly quality, channel assignment, routing metrics).
2. These individuals are then manipulated over many iterations using a predefined target function in order to converge to optimality.
3. The guiding principle is a probability function p_{xy}^{k} based on the relative weighting of pheromone intensity τ_{xy}^{α} and heuristic information η_{xy}^{β} which indicates the desirability or attractiveness of the option at a decision point.
4. At the end of each iteration, each individual adds pheromone τ_{xy} to its path. The amount of pheromone added is proportional to the quality of the solution (e.g. lower values of route delay receive more pheromone).

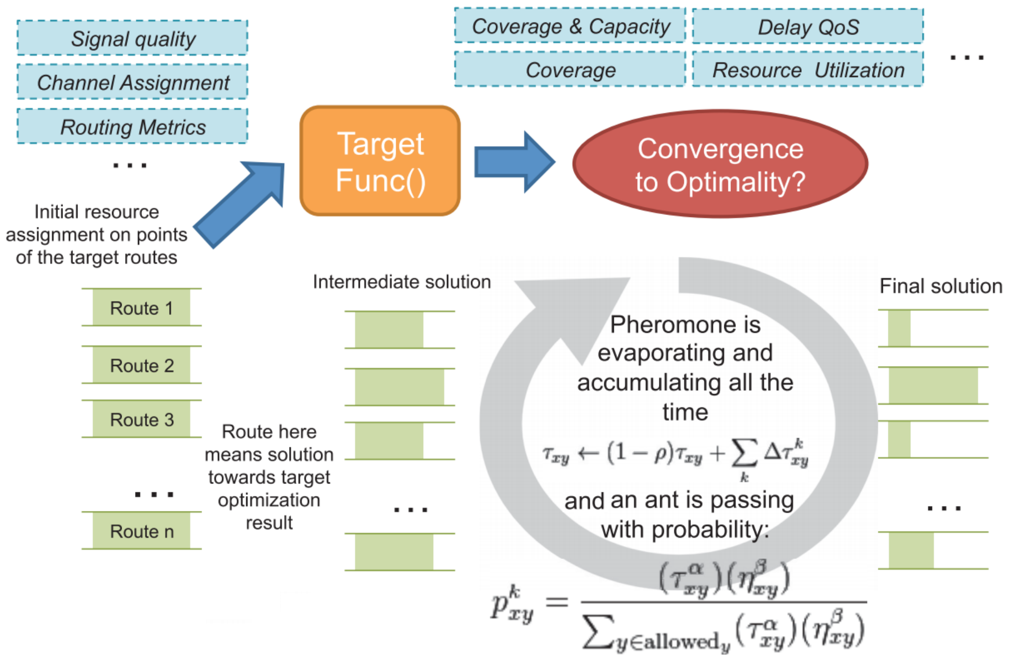
Signal quality
Channel Assignment
Routing Metrics
...
Initial resource assignment on points of the target routes
Target Func()
Coverage & Capacity
Delay QoS
Coverage
Resource Utilization
...
Convergence to Optimality?
Route 1
Route 2
Route 3
...
Route n
Route here means solution towards target optimization result
Intermediate solution
Final solution
Pheromone is evaporating and accumulating all the time
$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_k \Delta\tau_{xy}^k$
and an ant is passing with probability:
$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{y\in \text{allowed}_y}(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}$

Figure 3.8: Ant Colony Optimization (ACO) for HetNets [7]

The pseudocode of a simple ACO method can be found below (Algorithm 2).

```
begin
    initialize population;
    evaluate fitness of population;
    while not termination-condition do
        position each ant in a starting node;
        repeat
            foreach ant do
                choose next node by applying the state transition rule;
                apply step by step pheromone update;
            end
        until every ant has built a solution;
        evaluate fitness of population;
        update best solution;
        apply offline pheromone update;
    end
end
```

Algorithm 2: ACO pseudocode [20]

Besides ACO, Particle Swarm Optimization (PSO) is a well-known SI algorithm whose feasibility in Self-X functions has been recently studied. PSO is inspired by the social behaviour of some animal collectives who jointly perform actions in order to achieve a common goal (e.g. bird flocking). Each individual (i.e. particle) in a swarm behaves in a distributed way, using both its own intelligence and the collective intelligence of the swarm. Therefore, if one particle discovers an optimal path to reach a determined objective, the rest of the swarm will also be able to follow the good path instantly [20].

Regarding the current literature, SI techniques are found to be useful in the three self-X functionalities. A few examples are described next:

- **Self-Organization**: Network routing in HetNets can benefit from SI due to their inherent path optimization strategies. From these guidelines, the study in [27] presents an ACO algorithm to overcome the routing issues in HetNets by addressing a multi-objective routing optimization problem that uses network performance measures (e.g. delay, hop distance, cost).

- **Self-Healing**: As one representative work of cell outage compensation (COC), the study in [28] proposes an automatic particle swarm compensation algorithm for COC management scheme. The aim of COC is to mitigate the performance degradation induced by the outage by automatically adjusting the related radio parameters of the neighboring cells.

- **Self-Optimization**: The work presented in [29] proposes an ACO algorithm to overcome the problem of coverage optimization for dense deployments of small cells by finding the optimal pilot transmit power through the minimization of the cost function.

3.2.3 Artificial Neural Networks

ANNs are another computational method inspired from the biological neural networks of human brains. In the ANN, as illustrated in Figure 3.9, various artificial nodes (i.e. neurons) are interconnected to form a network of nodes, which processes information using a connectionist approach to computation. Each connection between neurons may have a numerical weight which can be tuned to make ANNs adaptive to inputs [20].

The neural models are often used to model complex relationships between inputs and outputs. Furthermore, ANNs do not require an exhaustive knowledge of the neural driving processes [30]. Therefore, ANNs have the innate ability to perform quite well in unsupervised environments, and thus their implementation in various HetNets problems has been discussed for estimating or approximating functions that depend on many unknown input conditions [26].

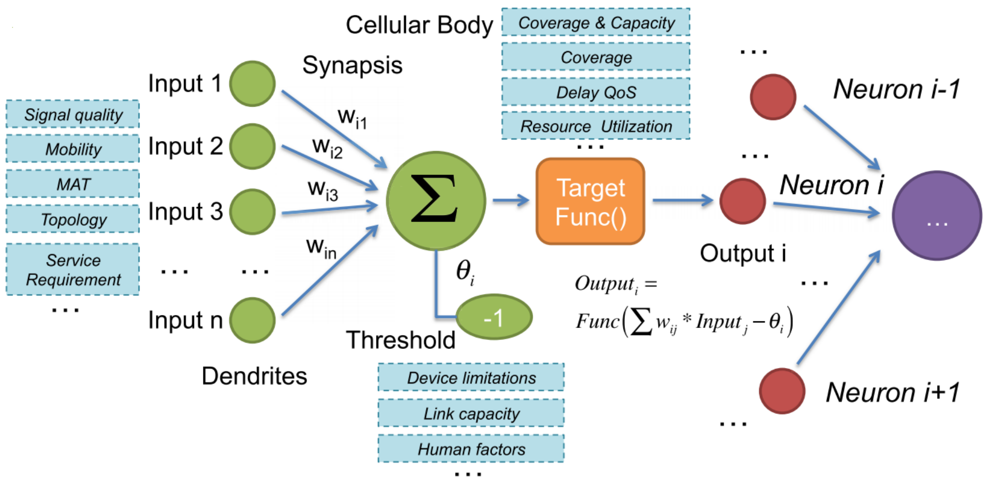

Figure 3.9: Artificial Neural Network (ANN) for HetNets [7]

Regarding ANNs and their relationship with Self-Optimization processes, the studies presented in [31] and [32] are well worth to mention. These works propose an ANN implementation to enhance the performance of the vertical handoff (Figure 3.10). With the upcoming 5G paradigm, there will be an urgent demand to develop efficient vertical handoff approaches that enable mobile terminals to seamlessly roam between the existent wireless networks. The proposed schemes enables the handoff user to adapt the destination network environment quickly and the variation of the throughput can be avoided efficiently.

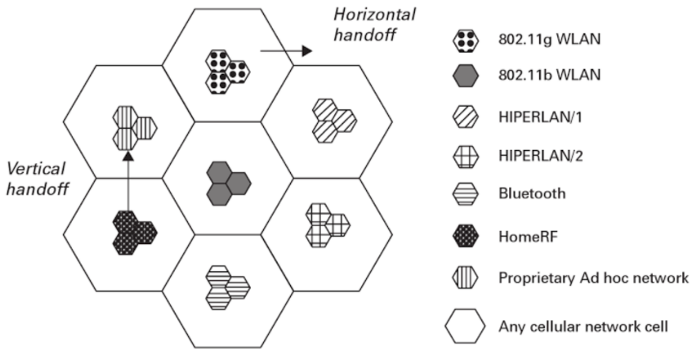

FIGURE 3.10: Illustration of vertical handoff [33]

3.3 Fuzzy Systems

"*Vagueness is a pervasive part of the human experience. Human language is an imprecise tool. Human perception is fraught with inaccuracy. Memories are fleeting and malleable. The real world is not an abstraction; it is not clearly perceived, well defined, and precisely calculated*" (Mark J. Wierman, 2010, p.53).

Fuzzy theory was developed to handle imprecise information. It starts with the concept of fuzzy set, whose function is to map (i.e. fuzzify) the set of input elements to a membership function which indicates the degree of truth belonging to the set.

$$\mu_A : X \to [0,1] \tag{3.4}$$

The degree of truth μ of a fuzzy set A that takes an input variable x ranges from 0 (i.e. x does not belong to A) to 1 (i.e. the other way around). Nevertheless, asides from that particular classic set where an element could belong or not, fuzzy logic allows the input variable to be mapped in a given set in a more broader sense (e.g. *no-barely-average-quite-yes* instead of *no-yes* degrees of truth). Humans do this kind of reasoning all the time, but it is a rather new concept for computers.

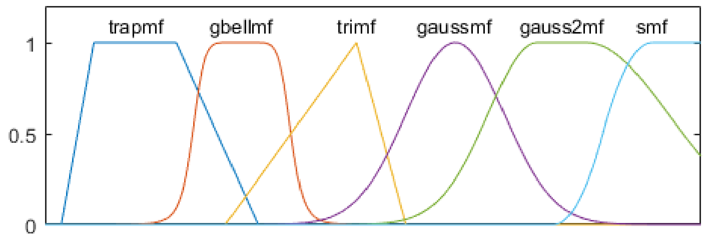

FIGURE 3.11: Membership function shapes (Matlab Fuzzy Logic Toolbox$^{\text{TM}}$)

Additionally, fuzzy logic allows the implementation of human knowledge in the form of if-then inference rules. A single fuzzy if-then rule has the following form:

If x is A, then y is B

where A and B are the linguistic values (e.g. low, medium and high) defined by the fuzzy sets X and Y, respectively. The linguistic input and output crisp values (e.g. signal quality and handoff decision) are x and y, respectively.

The if-part of the rule "x is A" is also known as the antecedent of the rule, while the then-part of the rule "y is B" is also called the consequent. For an if-then rule, the antecedent, p, implies the consequent, q. In binary logic, if p is true, then q is also true ($p \rightarrow q$). In fuzzy logic, however, if p is true to some degree of membership, then q is also true to the same degree [34].

Furthermore, it may be noticed that the human based rules in fuzzy logic might not be optimal and, therefore, optimization techniques should be performed in order to build up an accurate knowledge base.

Finally, the last step of a fuzzy inference process is the defuzzication, which is the method that determines a single crisp value from a fuzzy output set.

Fuzzy logic approach seems suitable to handle the imprecision of the practical wireless cellular networks [35]. Actually, fuzzy system techniques have been recently proposed to handle handoff decision algorithms. For instance, the study in [36] proposes a handoff decision algorithm based on type-2 fuzzy logic [3], which takes into account a variety of access network and user properties, and selects the network with the maximum satisfaction value. An illustration of fuzzy logic handoff in HetNets is shown in Figure 3.12.

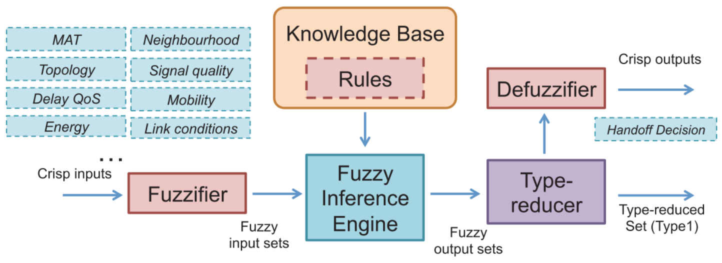

Figure 3.12: Fuzzy logic for HetNets [7]

Besides handoff-based applications, eICIC in HetNets can be successfully addressed by employing fuzzy logic, as studied in [37].

[3]For some applications, it is useful to define fuzzy sets in terms of more general forms of membership grade functions. An important form is $A : X \rightarrow L$, where L denotes a lattice. When L is a class of fuzzy numbers defined on [0,1], we obtain fuzzy sets of type-2 (Mark J. Wierman, 2010).

Chapter 4

Admission Control for Multi-tenant Radio Access Networks

One of the target goals to be addressed by future 5G architectures is the reduction of both capital and operational costs. The sharing of mobile network infrastructure among service providers (i.e. tenants) would allow the achievement of the expected requirements of the emerging HetNets, in terms of O/CAPEX, respectively. Additionally, the deployment of small cells under a multi-tenancy basis is another key component to be introduced in the future 5G networks. Within this framework, the implementation of neutral host models offering Small Cell as a Service (SCaaS) is regarded as an interesting approach to stimulate multi-operator small cells [38]. SCaaS models will enable cost savings through multi-operator deployments, while avoiding conflicts of interest within MNOs.

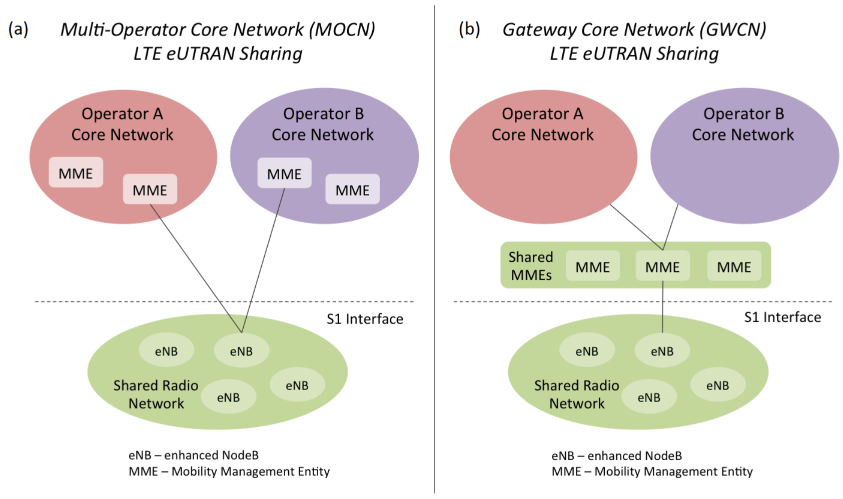

FIGURE 4.1: (a) Multi-Operator Core Network and (b) Gateway Core Network [39]

In order to provide network sharing functionalities in the well known legacy network infrastructures, two main architectures have been already proposed [39] by the 3rd Generation Partnership Project (3GPP). As illustrated in Figure 4.1, the Multi-Operator Core Network (MOCN) aims to directly connect a shared RAN to each of the multiple operator core networks, whereas the Gateway Core Network (GWCN) considers a shared core network instead. Note that the GWCN approach has its costs reduced compared with MOCN, but it has less flexibility and, therefore, it may reduce the level of differentiation amongst operators.

Admission Control, from now onwards denoted as AC, is in charge of either accepting or blocking new service requests. The split of radio resources among tenants at AC level seamlessly controls the acceptance of new bearers of each tenant, rather than scheduling the physical radio resources to packet flows on a per-tenant basis, so it does not require modifications at the lower layers of the protocol stack [40].

This chapter is basically based in the studies [40] [41], and it is structured as follows: section 4.1 introduces the main concepts of the proposed multi-tenant admission control, section 4.2 discusses the proposed algorithmic solution and section 4.3 presents the performance evaluation.

4.1 Multi-tenant Admission Control

Considering a multi-tenant RAN, where an infrastructure provider deploys its own network which is shared by multiple tenants, the sharing model and its technical and operational aspects will be detailed through Service Level Agreements (SLAs) between the infrastructure provider and each tenant.

The aforementioned RAN provides data transfer services for the exchange of information between the User Equipment (UE) and the mobile core network, namely known as Radio Access Bearers (RABs) in UMTS or Evolved-RAB (E-RAB) in LTE. Moreover, an End-to-End data service may have a certain QoS attributes (e.g. transfer delay, maximum bit rate or guaranteed bit rate) [42].

The AC function for multi-tenant RAN, which is executed at each involved cell, decides whether the establishment request of a new RAB is accepted or rejected. This decision should be made by considering three main factors: the overall resource utilisation in the cell, the QoS requirements of already active RABs and the requirements of a new RAB request.

The proposed multi-tenant AC algorithm, which will be detailed in the next section, is formulated based on the following main statements:

- As specified in the SLA terms, the capacity assigned to the tenant that request the RAB set-up has to be considered in the admission/rejection decision.

- The admission/rejection decision has to take into account the actual Resource Block (RB) utilisation as well. RBs determine the number of required radio resources at the physical layer and, due to the stochastic behaviour of the radio channel, they can only be statistically estimated. Therefore, in order to decide whether the capacity at the physical layer is enough to support the bit rate requirements of the new RAB, RB utilisation has to be considered.

4.2 Algorithmic Solution

The scenario assumed in this section consists in N cells labelled as $n = 1, ..., N$ shared by S tenants numbered as $s = 1, ..., S$. The AC must ensure that:

1. The amount of RBs required by the new RAB and by the already admitted RABs does not exceed the number of available RBs in the cell $\rho(n)$.

2. The available RBs are fairly shared amongst tenants.

Hence, the proposed multi-tenant AC will admit a new RAB if the next two conditions (A & B) are met simultaneously.

A. Capacity check at cell-level

This capacity check condition assesses whether the evaluated cell has sufficient physical resources which would allow the admission of a new RAB. This statement can be mathematically expressed as the following condition:

$$\sum_{s'=1}^{S} \rho_G(s', n) + \Delta\rho \leq \rho(n)\alpha_{th}(n) \tag{4.1}$$

where $\rho_G(s', n)$ is the average number of RBs of the n-th cell assigned to the RABs of the s-th tenant. $\Delta\rho$ is the estimated number of RBs required by the newly admitted RAB and is computed based on the required bit rate R_{req} and the estimated bit rate per RB $\hat{r}(n)$[1], respectively:

$$\Delta\rho = \frac{R_{req}}{\hat{r}(n)} \tag{4.2}$$

The last term $\rho(n)\alpha_{th}(n)$, which can be found in the right-side of the inequality, defines the cell-level AC threshold. It considers a fraction $\alpha_{th}(n) \in (0, 1]$ of the total number of RBs allocated in the n-th cell, leaving a margin to cope with handovers, for instance.

[1]It is left at the reader's choice to deepen on how this particular term is computed ([40], p.3)

B. Per-tenant capacity share check

This check sets an upper bound in the RBs used by the RABs of a tenant accordingly with the capacity agreed in the SLA. In this case, the capacity is defined by the Scenario Aggregated Guaranteed Bit Rate (SAGBR), which establishes the total bit rate to be guaranteed for all the RABs of a tenant.

Then, the nominal capacity share of a tenant s, $C(s)$, is defined as the ratio between the $SAGBR(s)$ across all the cells and the aggregated $SAGBR$ of all the tenants:

$$C(s) = \frac{SAGBR(s)}{\sum_{s'=1}^{S} SAGBR(s')} \tag{4.3}$$

From all the previous terms introduced, the per-tenant capacity share check condition can be now formulated as:

$$\rho_G(s,n) + \Delta\rho \leq \rho(n)\alpha_{th}(n) \cdot (C(s) + \Delta C(s,n)) \tag{4.4}$$

The above condition ensures that the s-th tenant will be allowed to use a fraction of the RBs in the n-th cell given by $C(s)$, plus an additional term $\Delta C(s,n)$ which considers the possible unused capacity left by the other tenants.

The term $\Delta C(s,n)$, which will be the key component regarding the optimization process carried out in Chapter 5, is defined as:

$$\Delta C(s,n) = \begin{cases} \Delta C_e(s,n) & \text{if } \Delta C_e(s,n) > 0 \\ \Delta C_b(s,n) & \text{if } \Delta C_e(s,n) = 0 \end{cases} \tag{4.5}$$

where $\Delta C_e(s,n)$[2] is the extra capacity which is potentially available for the s-th tenant in the n-th cell whenever the other $s' \neq s$ tenants leave unused capacity. Hence, the s-th tenant can get part of this extra capacity to serve a traffic load above the agreed capacity contracted through the SLA. The second term $\Delta C_b(s,n)$[2] ensures capacity share balance across all the cells and pursues fairness from a multi-cell perspective.

4.3 Performance Evaluation

The aforementioned AC approach has been evaluated in an outdoor urban micro scenario, in which each cell has one LTE carrier of 10 MHz (i.e. 50 RBs). The most significant downlink simulation parameters can be found in Table 4.1. Different offered loads of each tenant, denoted as T1 and T2 respectively, are simulated by varying the session arrival rate λ in each cell.

[2] The reader is invited to review how these terms are explicitly computed ([40], p.3)

Parameter	**Value**
ISD (Inter-Site Distance)	200 m
Path loss model	Urban micro-cell model with hexagonal layout
Shadowing standard deviation	3 dB in LOS and 4 dB in NLOS
Base station antenna gain	5 dB
Frequency	2.6 GHz
Transmitted power per RB	24 dBm
Number of RBs per cell $\rho(n)$	50 RBs (1 LTE carrier of 10 MHz)
UE Noise Figure	9 dB
R_{req}	1024 kb/s
Session duration	Exponential model, average 30 s
Session arrival rate	Poisson model with different simulated λ
$\alpha_{th}(n)$	1

TABLE 4.1: Simulation parameters

The capacities stipulated in the SLAs are SAGBR(1) = 25 Mb/s for Tenant 1 and SAGBR(2) = 37 Mb/s for Tenant 2, respectively. Specifically, the simulated scenario considers a total of $N = 2$ cells, where the total capacity per cell is estimated to be around 31 Mb/s. The nominal capacity shares of each tenant are C(1) = 40% for T1 and C(2) = 60% for T2.

Since the target objective of the proposed AC algorithm is to achieve an efficient utilization of the available RBs of each tenant, the performance evaluation will take into consideration different offered traffic load mixes in relation to tenant's capacity share. Since the efficiency in the resource usage is mainly provided by the term $\Delta C(s,n)$ (4.5), the performance assessment will consider, as a reference, the case in which $\Delta C(s,n)$ is set to 0.

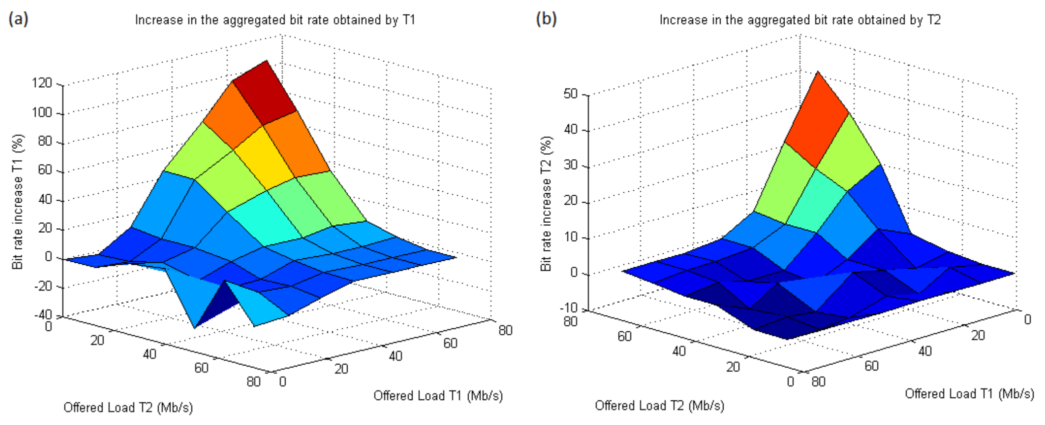

FIGURE 4.2: Increase in the aggregated bit rate obtained by (a) T1 and (b) T2, in relation to the reference case $\Delta C(s,n) = 0$

The gain achieved by the AC algorithm in the aggregated bit rate obtained by T1 and T2, in relation to the reference case where $\Delta C(s,n) = 0$, is illustrated in Figure 4.2. The proposed algorithm achieves an increase of the aggregated bit rate of T1 up to 106%, when the offered load of T2 is as low as 0 Mb/s. On the other hand, an increase of the aggregated bit rate of T2 up to 43% is achieved when the offered load of T1 is 0 Mb/s. It can be concluded that, whenever one tenant is not using all its capacity, the other one benefits from a higher aggregated bit rate, thus achieving a more efficient use of the radio resources.

Now, considering the following traffic mixes for T1 (Table 4.2) and T2 (Table 4.3), the performance experienced by each tenant is studied, in terms of aggregated bit rate and blocking probability.

		Tenant 1		
		Load Cell 1	**Load Cell 2**	**Total Load**
Traffic Mix	A	24.6 (H)	24.6 (H)	49.2 (H)
	B	19 (H)	6 (L)	25 (P)

Table 4.2: Selected traffic mixes for Tenant 1

		Tenant 2		
		Load Cell 1	**Load Cell 2**	**Total Load**
Traffic Mix	A	12.3 (L)	12.3 (L)	24.6 (L)
	B	12 (L)	25 (H)	37 (P)

Table 4.3: Selected traffic mixes for Tenant 2

The average offered load of a tenant in a given cell can be classified as: planned (denoted as P), well below the planned value (denoted as L) or well above the planned value (denoted as H). Note that the planned offered load refers to the one stipulated in the tenant's SLA. Similarly, the total offered load in both cells is denoted as P if it is equal to the respective tenant's SAGBR.

Traffic mix A describes a scenario in which the offered load of T1 is H in both cells, while the offered load of T2 is kept L. In this case, simulation results are depicted in Figure 4.3. It is observed that T1 can obtain a bit rate improvement of 33% with respect to the reference case, by allowing T1 to make use of the spare capacity of T2, while not considerably harming its bit rate. Moreover, there is a substantial reduction in the blocking probability for T1. Nevertheless, a slight degradation (below 2%) is noticed in the blocking probability for T2. Hence, in a neutral perspective, it can be concluded that T1 benefits far beyond in performance from what T2 loses.

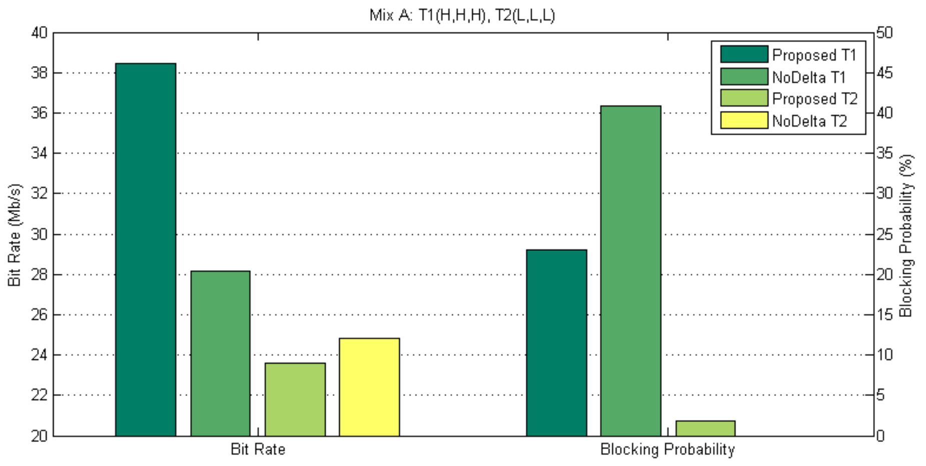

FIGURE 4.3: Aggregated bit rate and blocking probability obtained by each tenant with traffic mix A

On the other hand, in the traffic mix B, the total offered load of both tenants correspond to the planned one, but this load is not equally distributed along the two cells. In this case, simulations results are showcased in Figure 4.4 and Figure 4.5. It is observed that the proposed capacity share shift given by $\Delta C(s, n)$ allows T1 to efficiently handle the resource share across both cells. In this sense, T1 benefits from an increase of 18% in its aggregated bit rate and 8% for T2, with respect the reference case. Furthermore, an important reduction in terms of total blocking probability for both tenants is noticed. More specifically, T1 and T2 benefit from a blocking probability reduction of 70% and 64%, respectively.

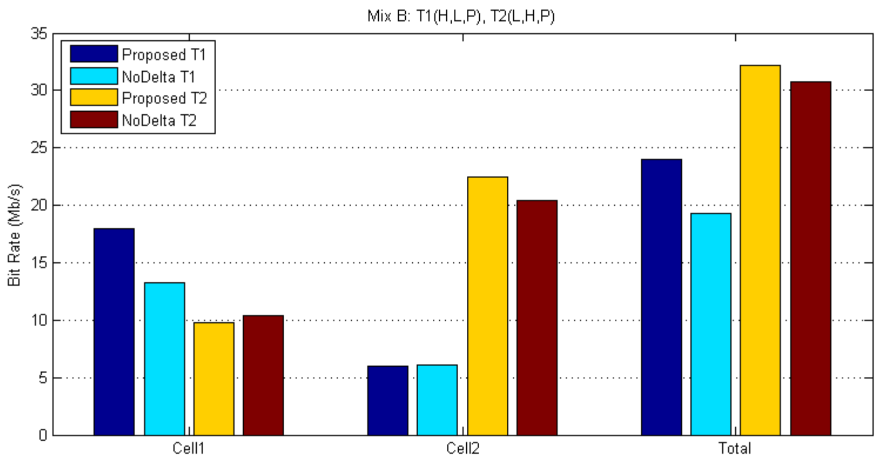

FIGURE 4.4: Bit rate obtained by each tenant in each cell and in the total scenario with traffic mix B

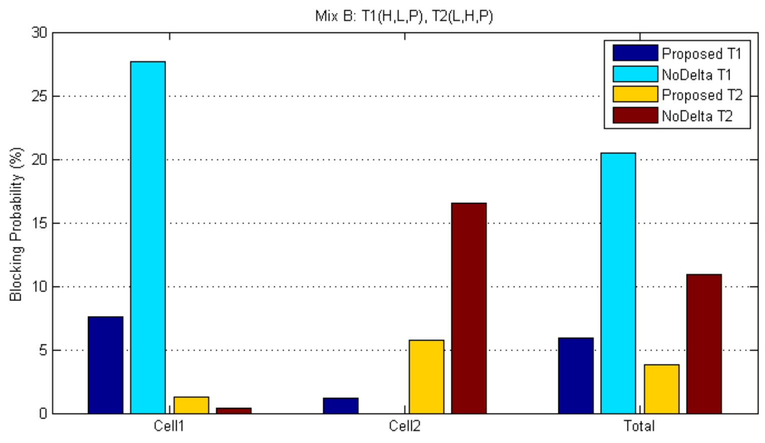

Figure 4.5: Blocking probability obtained by each tenant in each cell and in the total scenario with traffic mix B

Finally, Figure 4.6 illustrates the aggregated bit rate obtained by each tenant with the proposed algorithm and with the reference scheme, as a function of the total offered load of T2. The total offered load of T1 is kept constant at 49.2 Mb/s (corresponding to a H level of traffic mix A) and it is evenly distributed along the two cells. It can be noticed that, when the offered load of T2 is below its planned level of 24.6 Mb/s (considering the traffic mix A), T1 benefits substantially from the unused capacity left by T2. However, when the offered load of T2 is well above its planned level (i.e. 37 Mb/s), performance differences between both algorithms remain almost unnoticeable. In this last case, each tenant achieves a bit rate in accordance with is nominal capacity share $C(s)$. Moreover, the bit rate obtained by T2 is approximately the same with both schemes since T1 is not leaving unused capacity.

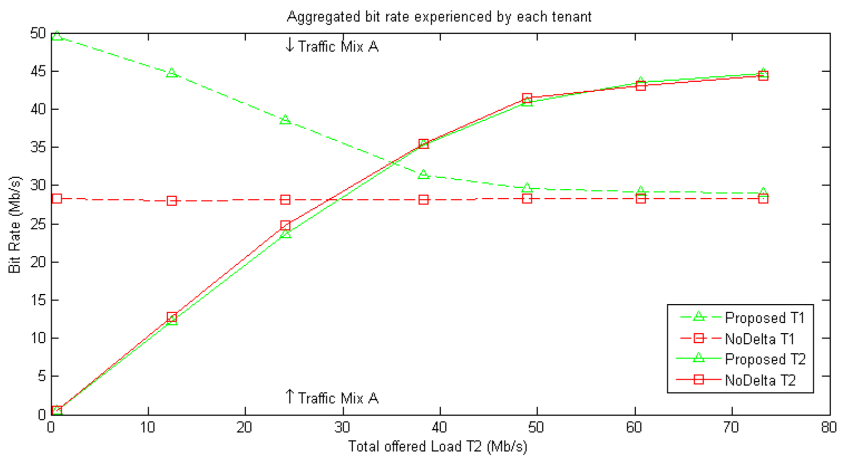

Figure 4.6: Aggregated bit rate experienced by each tenant

Chapter 5

Self-Organised Admission Control for Multi-tenant 5G Networks

In this chapter, a few AI techniques will be proposed in order to develop a self-organised AC for multi-tenant 5G networks. More specifically, the main objective is focused in self-learning the optimal value of the term $\Delta C(s, n)$, which was previously defined. The proposed methods correspond to the well known AI discipline of ML, in which two main learning algorithms arise:

- **Supervised learning**: primarily used when both input and output variables of a system are known, so that a mapping function can be learned ($Y = f(x)$). The main goal, however, is to approximate that mapping function in a way that output variables can be accurately predicted whenever new input data (i.e. unlearned data) is available.
- **Unsupervised learning**: unlike supervised learning, unsupervised learning lacks of any mapping function, since output data is either unknown or hard to obtain. Therefore, unsupervised algorithms are designed to discover the optimal structure or relationships between different input/outputs.

The structure of this chapter is summarized as follows: section 5.1 presents a supervised learning scheme which models the behaviour of $\Delta C(s, n)$ in the same network scenario used in the previous chapter. The subsequent section 5.2 proposes an unsupervised learning approach, whose aim is to self-optimize the value of $\Delta C(s, n)$ based on previous experience gained by interacting with the environment.

5.1 Supervised learning

The implementation of a supervised learning method for exploiting the knowledge of $\Delta C(s, n)$ has two main objectives. First of all, the knowledge itself allows the identification of the limits of $\Delta C(s, n)$ and its overall behaviour amongst different traffic load situations. This leads to the second goal, which aims to facilitate the implementation of the unsupervised learning AC method. Since there are 4 variables to self-optimize (i.e. $\Delta C(1,1)$, $\Delta C(1,2)$, $\Delta C(2,1)$ and $\Delta C(2,2)$), and the time to converge to the optimal solution could be excessively high, supervised learning will be applied in some $\Delta C(s,n)$ variables for simplicity's sake, and the left ones will be selected to be optimized.

For the scope of this work, an Adaptive Neuro-Fuzzy Inference System (ANFIS) is proposed as a supervised learning technique. ANFIS is a kind of ANN that incorporates a Takagi-Sugeno fuzzy inference engine, which only produces a single output after the defuzzification stage (i.e. one of the $\Delta C(s,n)$). On the other hand, 4 inputs are considered, corresponding to the offered loads of each tenant in each cell, taken directly from the environment/network simulator. Moreover, the fuzzy inference system (FIS) can incorporate two data clustering types: grid partition and subtractive clustering. The latest is considered, in which each input has as many membership functions as the number of clusters identified. As an example, 10 clusters are identified in Figure 5.1, although even more or less number of clusters could be found, considering the trade-off between training error and training time.

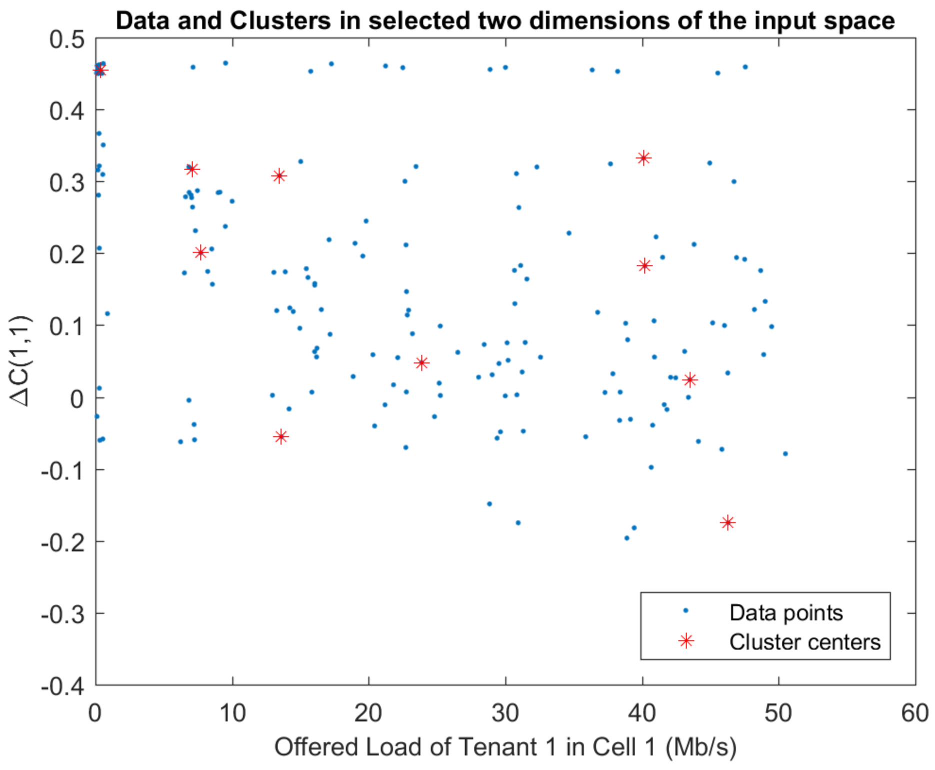

Figure 5.1: Subtractive clustering technique (Cluster's radius of influence = 0.65)

The overall proposed learning scheme is illustrated below (Figure 5.2).

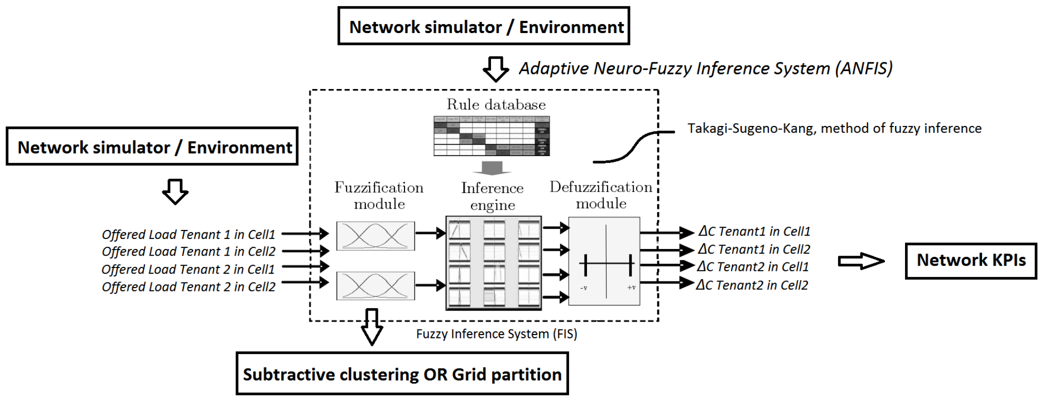
Network simulator / Environment
Adaptive Neuro-Fuzzy Inference System (ANFIS)
Rule database
Takagi-Sugeno-Kang, method of fuzzy inference
Network simulator / Environment
Fuzzification module
Inference engine
Defuzzification module
Offered Load Tenant 1 in Cell1
Offered Load Tenant 1 in Cell2
Offered Load Tenant 2 in Cell1
Offered Load Tenant 2 in Cell2
ΔC Tenant1 in Cell1
ΔC Tenant1 in Cell2
ΔC Tenant2 in Cell1
ΔC Tenant2 in Cell2
Network KPIs
Fuzzy Inference System (FIS)
Subtractive clustering OR Grid partition

FIGURE 5.2: Supervised learning scheme aimed to exploit $\Delta C(s, n)$ knowledge

Finally, once the input/output dataset has been trained through the aforementioned scheme, knowledge of $\Delta C(s, n)$ is available to precisely (>99%) exploit unlearned input data. A few representations of $\Delta C(s, n)$ as a function of different offered loads are shown in Figure 5.3.

The next step would be to retrieve the optimal value of $\Delta C(s, n)$ given a certain traffic load conditions in each optimization iteration of the unsupervised learning scheme presented in section 5.2, leaving a particular $\Delta C(s, n)$ to be self-optimized, just for the sake of simplicity.

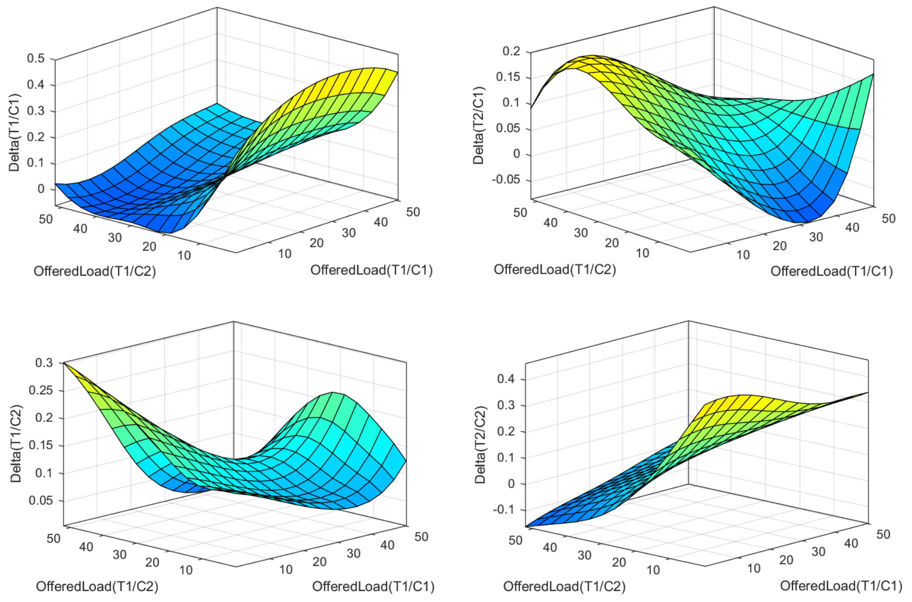

FIGURE 5.3: $\Delta C(s, n)$ as a function of T1 offered load (Mb/s) in both cells

5.2 Unsupervised learning (Fuzzy Q-learning)

In this section, an unsupervised learning algorithm will be implemented and simulated in the same LTE network seen throughout this work. Specifically, a fuzzy Q-learning algorithm is selected, which combines fuzzy logic with reinforcement learning.

In order to achieve self-optimization, each distributed agent must know what parameter tuning action should be performed in accordance to the current operation state. In the following lines, a brief review regarding the basics of fuzzy Q-learning is presented.

Shortly, Q-learning is a RL technique whose objective is to maximize a cumulative reward by taking actions in an environment. Q-learning builds up incrementally a Q-function, denoted as $Q(s, a)$, by estimating the discounted future rewards for taking actions a from given states s. A fuzzy version of Q-learning is considered in this work in order to inherit the benefits of fuzzy theory. Basically, fuzzy Q-learning allows to discretize the state and action spaces in order to avoid dealing with continuous and thus complex spaces.

The architecture of the self-optimization procedure is shown in Figure 5.4, which is clearly distributed. Besides the optimiser Q-Learning block, which updates the Q-function accordingly to the reward obtained, the fuzzy logic controller manages the set of environment states as its inputs (i.e. offered traffic loads and $\Delta C(s, 1)$) and the set of actions as its outputs (i.e. increment of $\Delta C(s, 1)$). Initially, it was planned to self-optimize both $\Delta C(s, 1)$, leaving $\Delta C(s, 2)$ to be optimized through supervised learning. Nevertheless, as the optimization time was incredibly high, it was decided to leave $\Delta C(1, 1)$ as the only variable to be self-optimized.

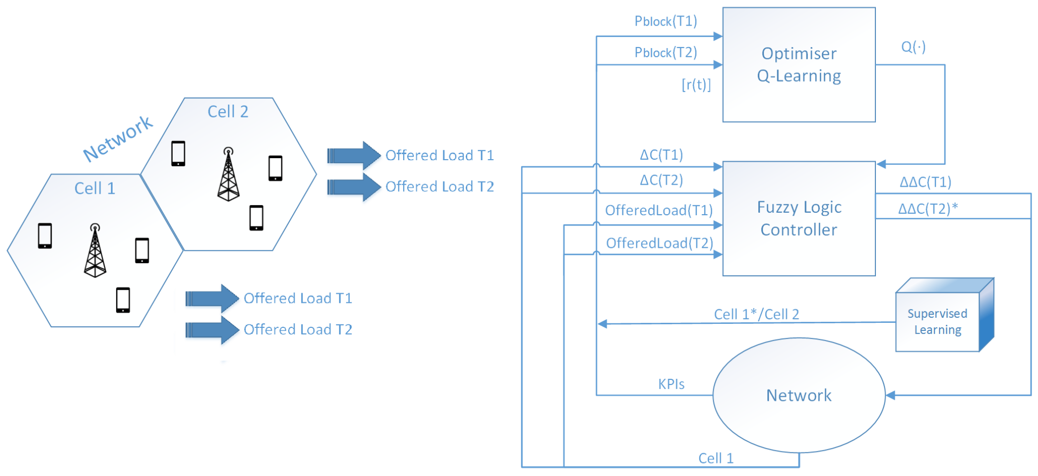

Figure 5.4: Architecture of the proposed self-optimization procedure

Next, the fuzzy Q-learning algorithm is presented with all its details.

First of all, let's define the concept of q-value. For each rule of the FIS, $a[i, j]$ is defined as jth action of rule i and $q[i, j]$ as its associated quality value (q-value). Therefore, the higher value of $q[i, j]$, the higher the trust for the corresponding tuning action selected.

To initialize the q-values in the algorithm, the following straightforward criteria is used:

$$q[i, j] = 0, 1 \leq i \leq N \text{ and } 1 \leq j \leq A \tag{5.1}$$

where $q[i, j]$ is the associated q-value to the rule i and action j. N is the total number of rules and A is the number of available actions per rule.

Now, for each activated rule (i.e. those with some non-zero degree of truth), an action is selected following an exploration/exploitation policy (e.g. ϵ-greed method). The agent should select the actions which produced highest rewards in the past. Nevertheless, the agent learns such action's performances by trying the actions that have not been selected before. Then, besides exploitation phase, an exploration policy should be considered in order to track the unexplored actions that yield maximum long term reward. In particular, the ϵ-greed method is defined as follows:

$$a_i = \begin{cases} random\{a_k, k = 1, 2, ..., A\}, & \text{with probability } \epsilon. \\ \text{argmax}_k \, q[i, k], & \text{with probability 1 - } \epsilon. \end{cases} \tag{5.2}$$

where a_i is the specific action for the rule i and ϵ is the learning rate of the exploration/exploitation policy. Usually, ϵ is not fixed along the optimization process. Instead, it progressively diminishes down to values close to zero, meaning that the exploration of potential actions decreases as well.

Then, the global action to be executed is determined by:

$$a(t) = \sum_{i=1}^{N} \alpha_i(s(t)) \cdot a_i(t) \tag{5.3}$$

where a is the parameter tuning action and $\alpha_i(s(t))$ is the activation function for the rule i. In other words, $\alpha_i(s(t))$ represents the degree of truth of an input state $s(t)$ in the t-th iteration:

$$\alpha_i(s(t)) = \prod_{i=1}^{M} \mu_{ij}(x_j(t)) \tag{5.4}$$

where M is the number of FIS inputs and $\mu_{ij}(x_j(t))$ is the membership function value for the j-th input and the i-th rule. For instance, considering the first rule where the four inputs are labelled as low (L), the activation function is given by:

$$\alpha_1(s(t)) = \mu_{11}(x_1(t)) \cdot \mu_{12}(x_2(t)) \cdot \mu_{13}(x_3(t)) \cdot \mu_{14}(x_4(t)) \tag{5.5}$$

The shapes of the membership functions are illustrated in Figure 5.5. For the offered traffic loads of both tenants, three gaussian membership functions are selected, labelled as Low (L), Medium (M) and High (H), respectively. For the $\Delta C(s,1)$, two edged trapezoidal and one triangular membership functions are used. Note that there are multiple options when choosing an appropriate shape of membership functions.

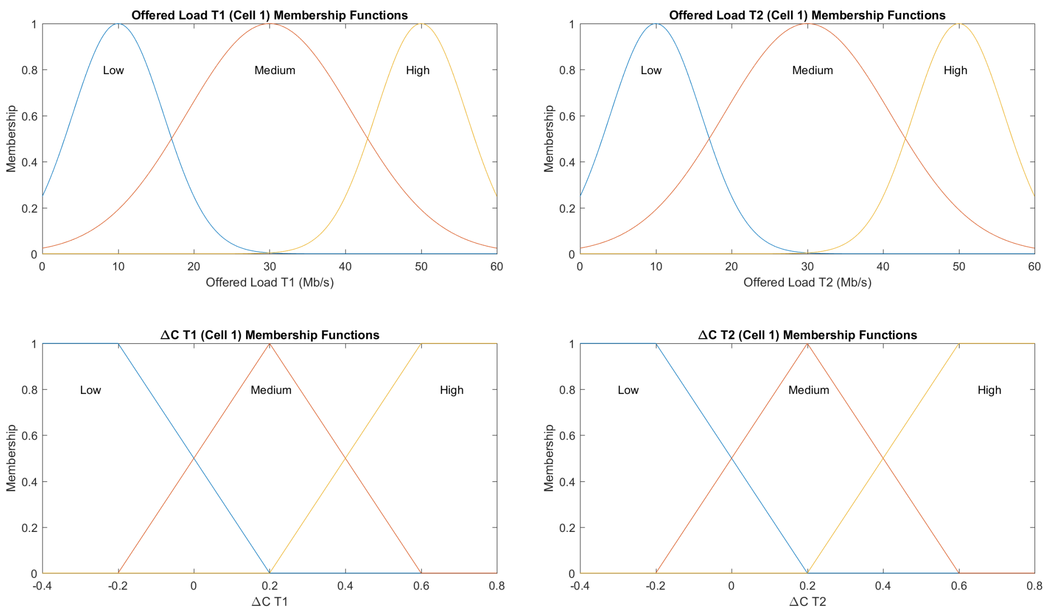

Figure 5.5: Fuzzy membership functions

The Q-function can be then calculated from the activation functions and the q-values of the different rules:

$$Q(s(t),a(t)) = \sum_{i=1}^{N} \alpha_i(s(t)) \cdot q[i,a_i] \tag{5.6}$$

where $Q(s(t),a(t))$ is the value of the Q-function for the state s and action a.

The next step involves leaving the system to evolve to the next state $s(t+1)$.

At this point, the reinforcement signal $r(t+1)$ is observed. In this work, the following reinforcement signal is considered, similarly as proposed in [43]:

$$r(t) = r_1(t) + r_2(t) + k1; \tag{5.7}$$

where $r(t)$ is the overall reinforcement signal itself, $r_1(t)$ and $r_2(t)$ are the reinforcement signal contributions of both tenants along the two cells, and $k1$ is a constant parameter. Specifically, $r_i(t)$ signals are computed as follows:

$$r_i(t) = k_2 \cdot \log(\frac{1}{(P_{block}(T_i) + k_3) \cdot 1000} + 1) \tag{5.8}$$

where k_2 and k_3 are constant parameters and $P_{block}(T_i)$ is the blocking probability of the T_i tenant in the whole scenario. The parameters used to compute out the reinforcement signal can be found in Table 5.1. Furthermore, an illustration of the reinforcement signal is shown in Figure 5.6. It can be observed that when the blocking probability of both tenants are zero, the reinforcement or reward obtained is maximum (i.e. equal to 1).

Parameter	Value
k_1	0.1357
k_2	100
k_3	0.1

TABLE 5.1: Reinforcement signal parameters

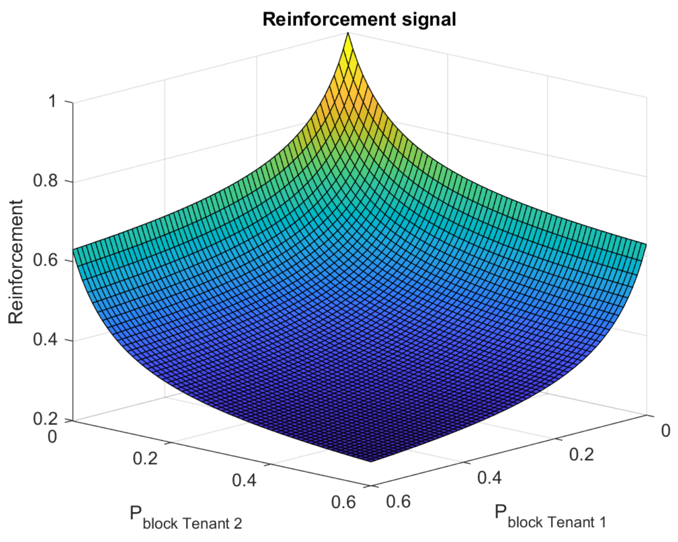

FIGURE 5.6: Reinforcement signal

Once the reinforcement signal of the next state $r(t+1)$ has been observed, the value of the new state denoted by $V_t(s(t+1))$ can be computed as:

$$V_t(s(t+1)) = \sum_{i=1}^{N} \alpha_i(s(t+1)) \cdot max_k q[i, a_k] \tag{5.9}$$

The error signal between consecutive Q-functions will be useful to update the q-values. It is given by:

$$\Delta Q = r(t+1) + \gamma V_t(s(t+1)) - Q(s(t), a(t)) \tag{5.10}$$

where ΔQ is the error signal, $r(t+1)$ is the reinforcement signal, γ is the discount factor and $Q(s(t), a(t))$ is the Q-function of the previous state. γ is set to 0.7, thus considering more importantly long term rewards.

Finally, the q-values can be updated by an ordinary gradient descent method:

$$q[i, a_i] = q[i, a_i] + \eta \cdot \Delta Q \cdot \alpha_i(s(t)) \tag{5.11}$$

where η is the learning rate, whose value is set to 0.5, meaning that older information is considered as important as newly one.

The aforementioned process is repeated from the action selection until the convergence is achieved.

A summary of the above described algorithm can be found below.

```
1. Initialize $q$-values:
$q[i,j] = 0, 1 \leq i \leq N$ and $1 \leq j \leq A$
2. Select an action for each activated rule ($\epsilon$-greedy policy):
$a_i = \begin{cases} random\{a_k, k = 1, 2, ..., A\}, & \text{with probability } \epsilon. \\ \operatorname{argmax}_k q[i,k], & \text{with probability } 1 - \epsilon. \end{cases}$
3. Calculate the global action:
$a(t) = \sum_{i=1}^{N} \alpha_i(s(t)) \cdot a_i(t)$
4. Approximate the $Q$-function from the current $q$-values and the degree of
  truth of the rules:
$Q(s(t), a(t)) = \sum_{i=1}^{N} \alpha_i(s(t)) \cdot q[i, a_i]$
5. Leave the system to evolve to the next state, $s(t+1)$.
6. Observe the reinforcement signal, $r(t+1)$, and compute the value of the
  new state denoted by $V_t(s(t+1))$:
$V_t(s(t+1)) = \sum_{i=1}^{N} \alpha_i(s(t+1)) \cdot max_k q[i, a_k]$
7. Calculate the error signal:
$\Delta Q = r(t+1) + \gamma V_t(s(t+1)) - Q(s(t), a(t))$
8. Update $q$-values by an ordinary gradient descent method:
$q[i, a_i] \leftarrow q[i, a_i] + \eta \cdot \Delta Q \cdot \alpha_i(s(t))$
9. Repeat the above described process starting from step 2. for the new
  current state until the convergence is achieved.
```

Algorithm 3: Fuzzy Q-learning algorithm [43]

The table below summarizes the main configuration and optimization parameters used in the proposed simulation scenario. As a reminder, the number of states corresponds to the total number of rules, and the actions ($\Delta C(s,n) = a + \Delta C(s,n)$) available for each rule are chosen to be as follows: one increment (+0.05), its homologous decrement (-0.05), and the no-change action (0).

Parameter	**Value**
Network Parameters	See Table 4.1
Number of states (i.e. rules)	3^4 (81)
Action space	[-0.05 0 +0.05]
Initial greed factor ϵ	0.9
Reducing rate of ϵ	1/650 x epoch
Discount factor γ	0.7
Learning rate η	0.5

TABLE 5.2: Optimization parameters

As observed in Figure 5.7, the exploration actions can be noticed as some of the reinforcement signals do not yield even near the maximum reward. Therefore, it is ensured that the whole state-action space is completely (or almost) checked.

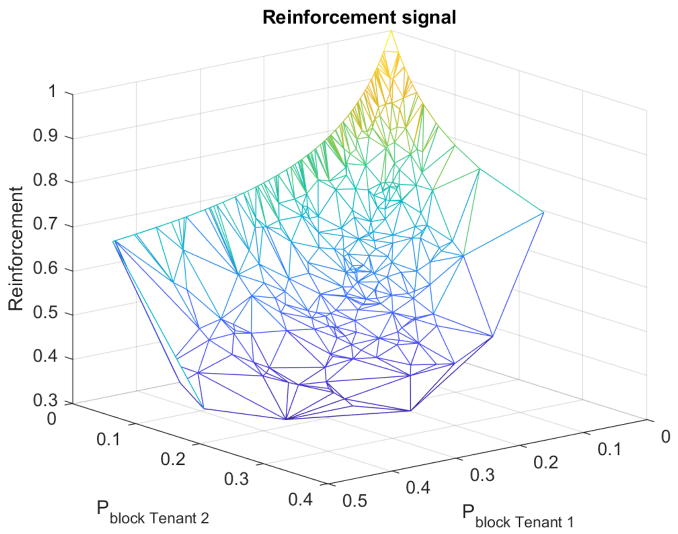

FIGURE 5.7: Simulated reinforcement signal after 500 epochs

The best consequent for each rule is determined by the highest action q-value. Table 5.3 shows three particular rules with three different actions.

Figure 5.8 illustrates how the best consequent for the 14th rule is selected. It can be observed that the highest q-value across the whole optimization process corresponds to the no-change action (i.e. 0), meaning that, in the mid-long term, the mentioned action will yield higher rewards. Regarding rules 32 (Figure 5.9) and 41 (Figure 5.10), the best actions to execute are the increase of $\Delta C(1,1)$ by 0.05 and vice versa, respectively.

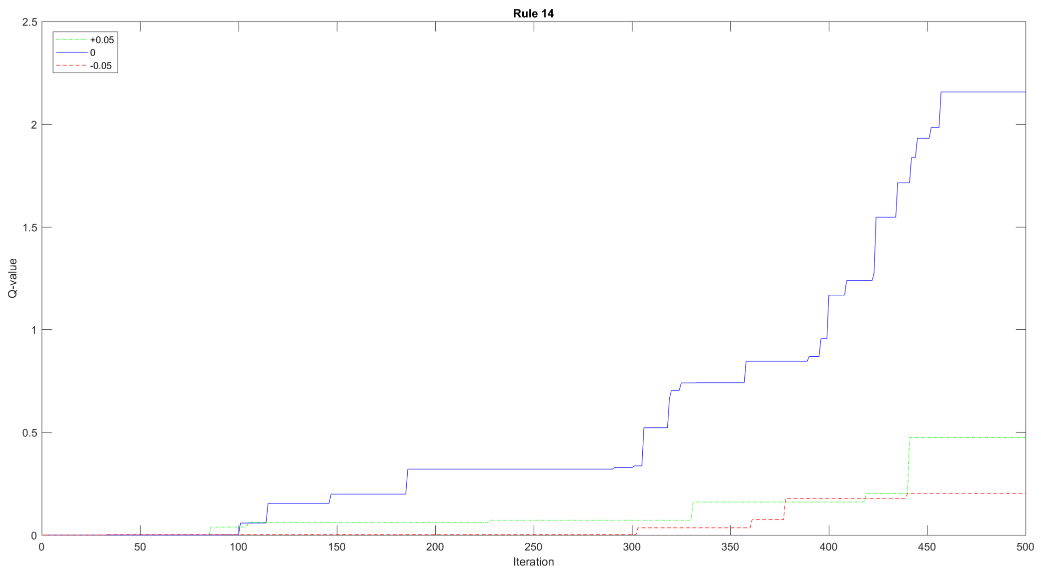

Figure 5.8: q-values evolution for rule 14

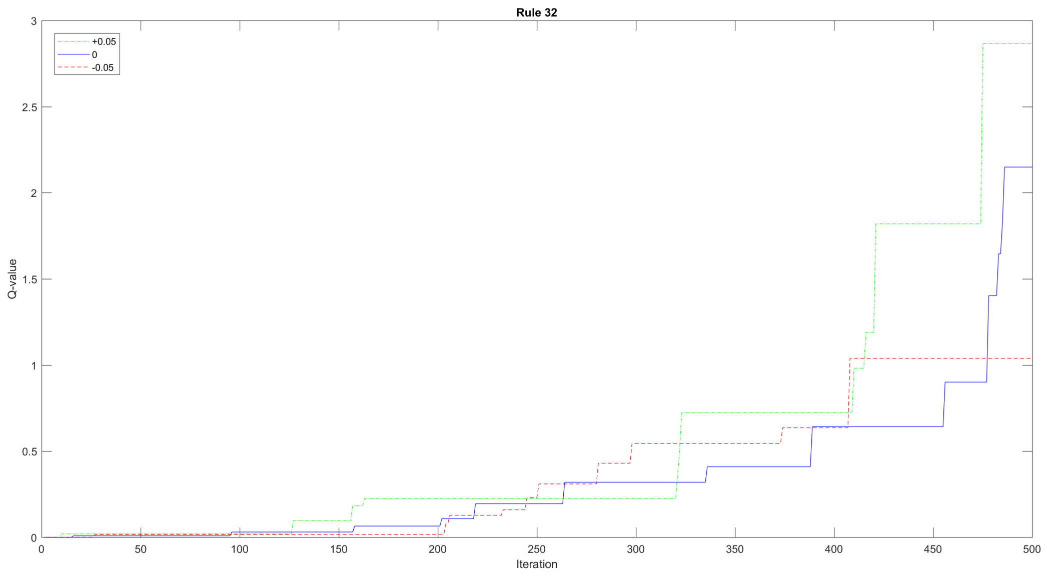

Figure 5.9: q-values evolution for rule 32

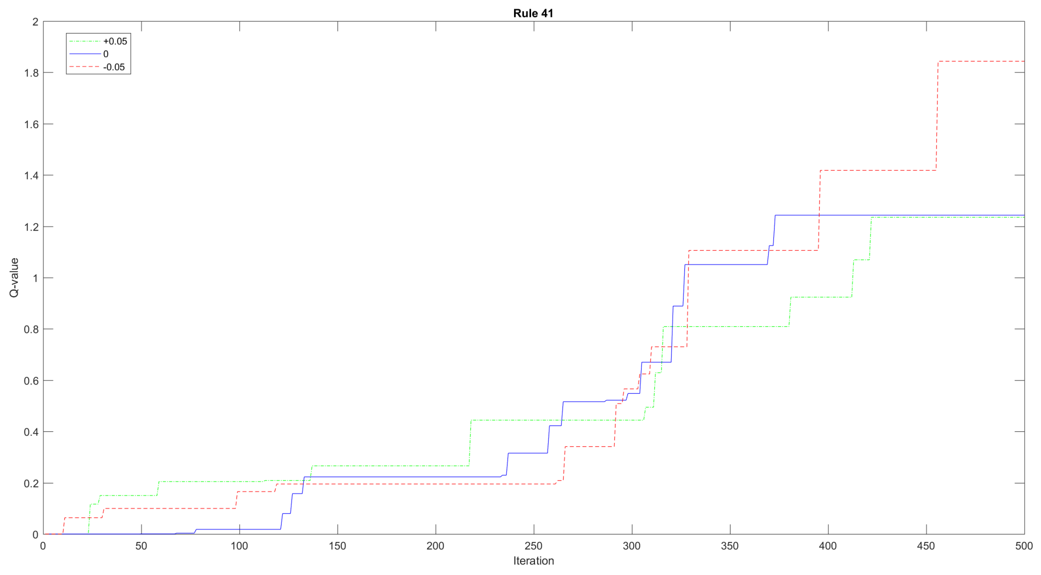

FIGURE 5.10: q-values evolution for rule 41

Rule	**Offered Load T1**	**Offered Load T2**	$\Delta C(1,1)$	$\Delta C(2,1)$	**Candidate actions**	**Best action**
14	L	M	M	M	[-0.05 0 +0.05]	0
32	M	L	M	M	[-0.05 0 +0.05]	+0.05
41	M	M	M	M	[-0.05 0 +0.05]	-0.05

TABLE 5.3: Fuzzy inference rule base acquired by Q-learning

Once the fuzzy inference rule base acquired by the proposed algorithm is built, the network performance can be evaluated. In this particular case, the blocking probability of each tenant and each cell is selected as a network performance measurement. Additionally, the results given by the proposed fuzzy Q-learning algorithm are compared with the reference case in which $\Delta(s, n)$ is fixed to 0 (denoted as 'NoDelta' case).

Figure 5.11 shows the blocking probability per cell and tenant in the exploitation/exploration phase. It is observed that substantial improvements are achieved by Fuzzy Q-learning approach with respect to the fixed configuration ('NoDelta'), especially in the T1 domain. Furthermore, Figure 5.12 illustrates the differences between fully exploiting the system (fixed $\epsilon = 0$) and using a exploitation/exploration trade-off (initial $\epsilon = 0.9$, with a decreasing rate of $1/650$ per epoch). As expected, the network performance is slightly better when exploration is not taken into account. Nevertheless, this work considers any potential action which could yield higher rewards in the future, hence the second approach applies.

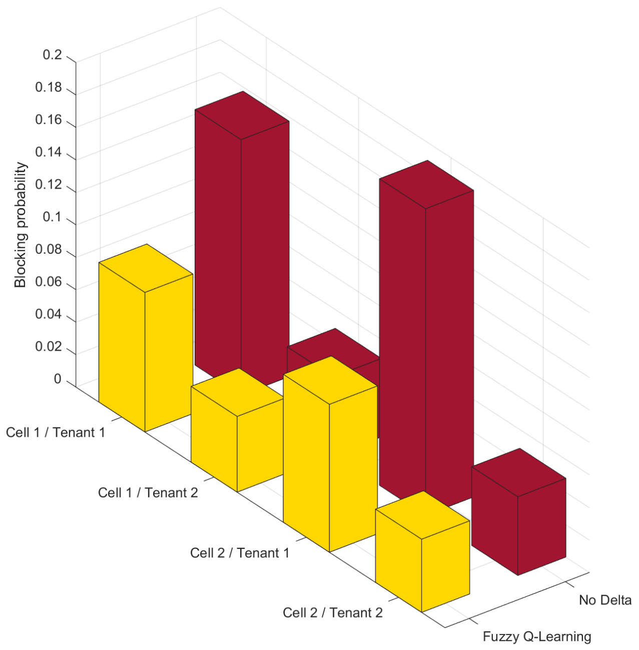

Figure 5.11: Blocking probability per cell and tenant in the exploitation/exploration phase (initial greed factor $\epsilon = 0.9$)

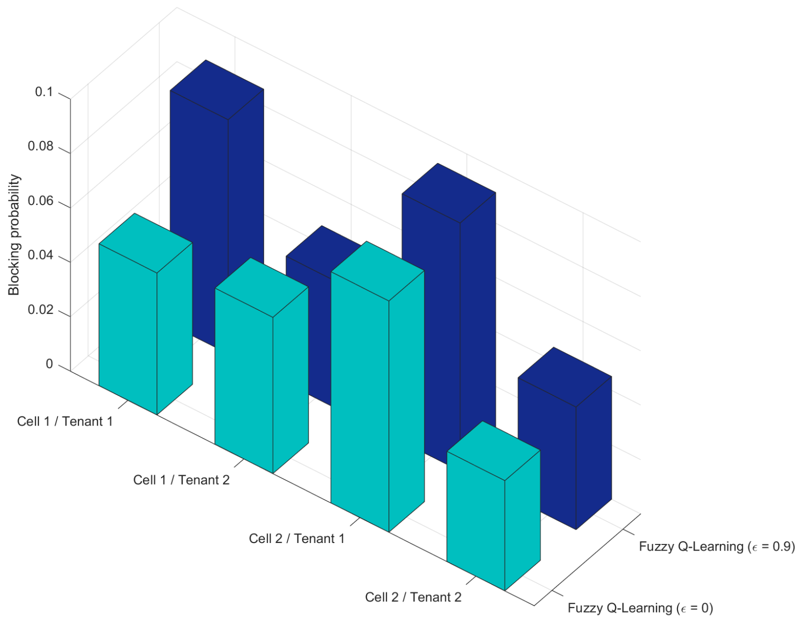

Figure 5.12: Blocking probability per cell in the exploitation (fixed greed factor $\epsilon = 0$) and exploitation/exploration phase (initial greed factor $\epsilon = 0.9$)

Finally, the exact values of the simulation, for the cell 1 and cell 2, are shown in Table 5.4 and Table 5.5, respectively.

		Blocking Probability				
		NoDelta	$\Longrightarrow$	**FQL** ($\epsilon = 0.9$)	$\Longrightarrow$	**FQL** ($\epsilon = 0$)
Cell 1	**T1**	0.157	+45.2%	0.0860	+39.4%	0.0521
	T2	0.0483	+3.5%	0.0466	-1.5%	0.0473

Table 5.4: Blocking probability (Cell 1) in the reference case, exploitation/exploration (greed factor $\epsilon = 0.9$) and exploitation phase (fixed greed factor $\epsilon = 0$)

		Blocking Probability				
		NoDelta	$\Longrightarrow$	**FQL** ($\epsilon = 0.9$)	$\Longrightarrow$	**FQL** ($\epsilon = 0$)
Cell 2	**T1**	0.188	+51.5%	0.0911	+6.8%	0.0849
	T2	0.0485	+7%	0.0451	+10.1%	0.0405

Table 5.5: Blocking probability (Cell 2) in the reference case, exploitation/exploration (greed factor $\epsilon = 0.9$) and exploitation phase (fixed greed factor $\epsilon = 0$)

Chapter 6

Conclusions and Future Work

The *statu quo* of mobile networks has been shifted from classical centralized network architectures to distributed heterogeneous networks with a higher degree of automation, cooperation and intelligence. The tight requirements of future 5G networks, in terms of reduced latency and increased capacity, has accelerated the introduction of self-organizing networks, whose automated mechanisms will address the seamlessly configuration, optimization and reposition of wireless networks. Moreover, AI-based techniques are seen as an excellent opportunity, with still a large room of improvement, to build up an intelligence system able to help the emerging HetNets to reach the aforementioned stringent 5G requirements.

Besides AI-related SON techniques, the inclusion of softwarization technologies such as Software-Defined Networks (SDNs) will substantially change the way how 5G networks will be managed [44]. Software network technologies, as illustrated in Figure 6.1, are aimed to be fundamental enablers to fulfill the requirements of programmability (e.g. service agility, service diversity and resource efficiency), flexibility (e.g. re-configurability, reusability and infrastructure sharing), adaptability (e.g. self-configuration, self-healing and self-optimization) and capabilities (e.g. mobile edge computing, network slicing, autonomic network management) expected to be inherent in 5G networks.

The potential benefits of softwarization in 5G include O/CAPEX reduction, shorter times for service creation and service adaptation, efficient service lifecycle management, energy consumption reduction towards a sustainable green networks, and improved quality of experience for users, among others. Indeed, SDNs are envisaged as one of the key features of 5G networks as they will drive the paradigm shift of mobile network design and implementation. Expectations are that a number of enablers would be required such as multi-tenancy management, multi-domain orchestration, end-to-end network slicing, uniform virtualization and abstraction facilities [45].

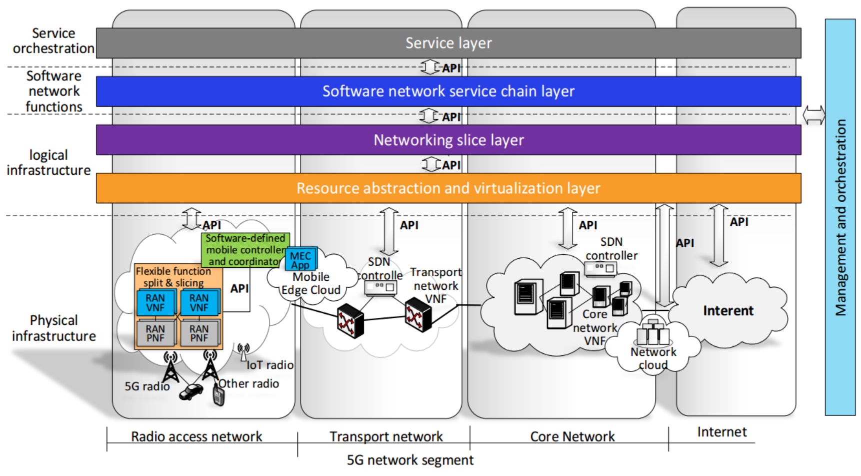

Figure 6.1: Software network technologies in 5G overall architecture [45]

Throughout this work, the state of the art of the most promising artificial intelligence-based techniques for the emerging HetNets has been carefully reviewed. Furthermore, the applicability and feasibility of each of them in every Self-X function has been assessed. One of the target objectives of the present work has been the study of QoS optimization for the future HetNets. More specifically, the admission control for multi-tenant RANs was the chosen topic to start exploring novel AI approaches to self-optimize the appropriate AC parameters. Hence, an accurate study of the self-optimised AC strategy for adjusting the share of resources used by each tenant was required in order to introduce the proposed AI algorithm. Amongst the all possible candidate AI solutions, a fuzzy Q-learning algorithm has been selected for the self-optimization process, as its model-free approach allows to build up an optimal action-selection policy, without actually requiring any network environment knowledge. Other AI options such as bio-inspired algorithms or artificial neural networks could have been considered as well. A simulation-based analysis of the proposed reinforcement learning algorithm has been presented to assess the potential improvements achieved by each tenant in each cell with respect to a baseline scheme. As for the results, the reduction achieved in the blocking probability by the proposed fuzzy Q-learning algorithm, in relation to the reference case where $\Delta C(s, n) = 0$, when considering an exploitation/exploration policy, has been 45.2% and 51.5% for T1 in cell 1 and cell 2, respectively. Nevertheless, T2 slightly benefits from the proposed approach (improvement of 3.5% and 7%, respectively), since its offered load was below its SAGBR, thus allowing T1 to efficiently handle the resource share across both cells and consequently benefit from substantial blocking probability reduction. Finally, the performance when considering a fully exploiting system (i.e. $\epsilon = 0$) has been evaluated. Despite improving the performance (up to 39.4%) in relation to the exploitation/exploration policy, it does not consider potential actions which could yield higher rewards in the long term future, thus the latest approach (i.e. $\epsilon = 0.9$) prevails in this work.

Appendices

Appendix A

Matlab® source code

Network Simulator (script sim_AC_vX.m) [40][41]

```
clear;

vector_variation=[0.2,0.4,0.6,0.8,1.0,1.2];

for loop_variable=vector_variation
fprintf('Sim for param: %f \n',loop_variable);

rng(740);        %random number generation seed

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%INPUT PARAMETERS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%CONSTANTS:
config.NO_SLICING=0;  %The AC only accounts for the 1st check (global check).
config.SLICING_NO_DELTA=1;  %The AC does not account for the delta
config.SLICING_DELTA=2;     %The AC account for delta parameters

%config.AC_algorithm=config.NO_SLICING;
%config.AC_algorithm=config.SLICING_NO_DELTA;
config.AC_algorithm=config.SLICING_DELTA;

%General scenario parameters
config.ISD=200;  %m IntersiteDistance
config.cell_R=(config.ISD/2)*2/sqrt(3);   %Cell Radius
config.num_cells=2;
config.num_tenants=2;
config.time_step=0.1;      %Duration of the simulation time step in s.
config.simulation_duration=50000.0;    %Simulation duration in s.

config.num_RBs=50;   %Number of RBs (default for all the cells)
config.B_RB=180; %Bandwidth of one RB in kHz
config.Ptot=41; %dBm. Total power per LTE carrier (default for all the cells)
config.P_RB=config.Ptot-10.0*log10(config.num_RBs);  % dBm.  Power per RB.
config.antenna_gain=5.0;
%config.PIRE_RB=config.P_RB+config.antenna_gain;      %dBm
config.noise_figure_UE=9;   %dB.
config.Pnoise_RB=-174+config.noise_figure_UE+10*log10(config.B_RB*1E3);    %
    Noise power per RB.
config.Pnoise_RB=power(10,0.1*config.Pnoise_RB);  %mW.  Noise power per RB.

%Propagation model parameters
config.prop_model_params.height_BS=10;   %meters
config.prop_model_params.height_UE=1.5;
config.prop_model_params.f=2.6;  %GHz
config.prop_model_params.d_BP=4*(config.prop_model_params.height_BS-1)*(config
    .prop_model_params.height_UE-1)*config.prop_model_params.f*1E9/3E8;
config.prop_model_params.dmin=10;      %minimum distance between a UE and a
    cell.
config.prop_model_params.sigma_LOS=3;      %dB
config.prop_model_params.sigma_NLOS=4;     %dB

%Spectral Efficiency computation parameters
config.spec_eff_params.SINRmin=-10.0;  % dB
config.spec_eff_params.SINRmin=power(10,0.1*config.spec_eff_params.SINRmin); %
    linear units
config.spec_eff_params.alfa=0.6;
config.spec_eff_params.Smax=4.4;  % b/s/Hz
config.spec_eff_params.SINRmax=power(2,config.spec_eff_params.Smax/config.
    spec_eff_params.alfa)-1;  %linear units

%Traffic parameters (default values)
config.traffic_params.Rbreq=512*2;     %kb/s Required bit rate of each session
     (default value)
config.traffic_params.duration=10;  %seconds. Average duration of each session
    . (default value)
config.traffic_params.lambda=1.0;    %Sessions/sec. Session generation rate (
    default value)

%Admission parameters
config.admission_params.alfa_th=0.9; %Admission threshold (default)
config.admission_params.beta=1.0;
config.admission_params.gamma=1.0;
config.admission_params.Cextra_min=0.0;

%config.time_window_utilisation_averaging=10.0;     %s Time window for
    averaging the RB utilisation
config.time_window_utilisation_averaging=0;     %s Time window for averaging
    the RB utilisation
config.time_window_utilisation_averaging_samples=config.
    time_window_utilisation_averaging/config.time_step;

config.time_window_delta_averaging=300.0;     %s Time window for averaging the
    delta parameters of the AC.
config.time_window_delta_averaging_samples=config.time_window_delta_averaging/
    config.time_step;

config.time_window_bit_rate_averaging=30.0;      %s Time window for estimating
    the bit rate per RB
config.time_window_bit_rate_averaging_samples=config.
    time_window_bit_rate_averaging/config.time_step;

%Capacity share parameters per tenant:
config.C(1)=0.4;
config.C(2)=0.6;
config.C_avg_multi_cell=zeros(config.num_tenants,1);
config.C_avg_multi_cell_samples=zeros(1+config.simulation_duration/config.
    time_step,config.num_tenants);
config.aggregate_avg_Rb_multi_cell=zeros(config.num_tenants,1);  %Measures the
     aggregate avg Rb of each tenant in the whole scenario
stats.num_adm_above_global_SAGBR=zeros(1,config.num_tenants);  %Measures the
    number of admissions above the global SAGBR of the whole scenario
stats.num_rej_below_global_SAGBR=zeros(1,config.num_tenants);  %Measures the
    number of rejections below the global SAGBR of the whole scenario

config.Cell_Capacity_theoretical=config.num_RBs*config.B_RB*config.
```

```
    spec_eff_params.Smax; %In kb/s
config.Cell_Capacity_effective=config.Cell_Capacity_theoretical*0.7757;     %
    Empirical correction (for accounting blocking 2%)

config.correction_factor=0.7757;   %Parameter Theta of the algorithm  (Factor
    equal to the ratio Effective Capacity/TotalCapacity, where Effective
    capacity reflects the maximum offered load for a max. blocking probability
     while TotalCapacity reflects the max capacity of a cell based on the
    amount of RBs).

%SAGBR for the total scenario
config.SAGBR(1)=config.num_cells*config.C(1)*config.Cell_Capacity_effective;
config.SAGBR(2)=config.num_cells*config.C(2)*config.Cell_Capacity_effective;

%SAGBR "per cell"
config.cellSAGBR(1)=config.C(1)*config.Cell_Capacity_effective;
config.cellSAGBR(2)=config.C(2)*config.Cell_Capacity_effective;

%Distribute and initialize the cells:
if config.num_cells>19
    fprintf('ERROR (num_cells exceeds 19: not supported)!!!!!\n');
end;

for n=1:config.num_cells
    BS(n)=base;
    BS(n).id=n;
    BS(n).init_BS(config);
end;

%If we want to have different traffic parameters per cell/tenant, specify
%them here (otherwise the traffic parameters are set to the default
%values).

%BS(1).lambda(1)=1.0*loop_variable;
%BS(1).lambda(1)=0.6;
%BS(1).lambda(2)=0.4;

BS(1).lambda_ini(1)=loop_variable;
%BS(1).delta_lambda(1)=0.2;
%BS(1).delta_lambda(1)=0;
%BS(1).freq_traffic_period(1)=1/(1*3600); %1h for this example.
%BS(1).time_shift(1)=0;

BS(1).lambda_ini(2)=0.2;
%BS(1).delta_lambda(2)=0.2;
%BS(1).delta_lambda(2)=0;
%BS(1).freq_traffic_period(2)=1/(1*3600); %1h for this example.
%BS(1).time_shift(2)=0.5*3600;   %0.5h for this example.

BS(1).lambda(1)=max(BS(1).lambda_ini(1)+BS(1).delta_lambda(1)*cos(2*pi*BS(1).
    freq_traffic_period(1)*(0-BS(1).time_shift(1))),1E-8);
BS(1).lambda(2)=max(BS(1).lambda_ini(2)+BS(1).delta_lambda(2)*cos(2*pi*BS(1).
    freq_traffic_period(2)*(0-BS(1).time_shift(2))),1E-8);

BS(2).lambda_ini(1)=loop_variable;
BS(2).lambda_ini(2)=0.2;
BS(2).lambda(1)=max(BS(2).lambda_ini(1)+BS(2).delta_lambda(1)*cos(2*pi*BS(2).
    freq_traffic_period(1)*(0-BS(2).time_shift(1))),1E-8);
BS(2).lambda(2)=max(BS(2).lambda_ini(2)+BS(2).delta_lambda(2)*cos(2*pi*BS(2).
    freq_traffic_period(2)*(0-BS(2).time_shift(2))),1E-8);

BS(1).duration(1)=30.0;
BS(1).duration(2)=30.0;
BS(2).duration(1)=30.0;
BS(2).duration(2)=30.0;

BS(1).alfa_th=1.0;
BS(2).alfa_th=1.0;

%If we want to have different frequencies per cell, specify them here.
%Otherwise, by default all the cells use the same frequency
BS(1).freq_index=1;
BS(2).freq_index=2;
%BS(3).freq_index=3;

%After having modified cell-specific parameters, initialize the radio
%parameters of each cell and the next session arrival rates:

%Schedule the arrival of the first session of each tenant in each cell.
for n=1:config.num_cells
    BS(n).init_radio_and_next_arrivals(config);
end;

%Main simulation
t_index=0;

%preallocate matrix for the logs:
arrival_log=zeros(5E5,200);

num_entries=0;  %index to register the simulation_log

for time=0:config.time_step:config.simulation_duration
    if mod(time,10)==0
        fprintf('Simulating time: %f \n',time);
    end;

    changing_conditions=0;  %To identify if there is some change (arrival/end)
         in the time step.

    t_index=t_index+1;

    %Check session finalisations:
    for n=1:config.num_cells
        for s=1:config.num_tenants
            if BS(n).numUEs(s)>0
                end_process=0;
                i=1;
            else
                end_process=1;
            end;
            while ~end_process
                if BS(n).UElist{s}(i).end_session_time<=time
                    %Remove UE i from the list.
                    BS(n).UElist{s}=horzcat(BS(n).UElist{s}(1:i-1),BS(n).
                        UElist{s}(i+1:BS(n).numUEs(s)));
                    BS(n).numUEs(s)=BS(n).numUEs(s)-1;

                    changing_conditions=1;

                    %Note: the next UE to check is still the index i!!!
                    %(because we have shifted the UEs in the array)
                    %Then, we only increase i when we do not remove the UE.
                else
                    i=i+1;
                end;
                if i>BS(n).numUEs(s)
```

```
                    end_process=1;
                end;
            end;
        end;
    end;

    %Check session starts:
    for n=1:config.num_cells
        for s=1:config.num_tenants
            %Compute lambda
            BS(n).lambda(s)=max(BS(n).lambda_ini(s)+BS(n).delta_lambda(s)*cos
                (2*pi*BS(n).freq_traffic_period(s)*(time-BS(n).time_shift(s)))
                ,1E-8);

            %if BS(n).time_next_session_arrival(s)<=time
            while BS(n).time_next_session_arrival(s)<=time  %By putting the
                while, we allow multiple arrivals in a time step.
                %New session
                %First, execute the admission process (assuming it is
                    independent of the
                %UE position):
                BS(n).num_session_attempts(s)=BS(n).num_session_attempts(s)+1;

                %Compute duration (even if it is not admitted later on)
                duration_session=(-BS(n).duration(s))*log(1-rand());

                BS(n).offered_load(s)=BS(n).offered_load(s)+duration_session*
                    BS(n).Rbreq(s);

                admit=BS(n).admission(s,BS(n).Rbreq(s),config);

                if admit
                    %Generate the new UE of tenant s
                    BS(n).numUEs(s)=BS(n).numUEs(s)+1;
                    BS(n).UElist{s}(BS(n).numUEs(s))=UE;
                    BS(n).UElist{s}(BS(n).numUEs(s)).init_UE(config,BS,n,s);

                    BS(n).UElist{s}(BS(n).numUEs(s)).Rbreq=BS(n).Rbreq(s);

                    %Compute the end session time for this UE:
                    BS(n).UElist{s}(BS(n).numUEs(s)).end_session_time=time+
                        duration_session;

                    %Compute statistics:
                    if (BS(n).avg_bit_rate_assigned_per_tenant(s)+BS(n).Rbreq(
                        s))>config.cellSAGBR(s)
                        %Measure of SAGBR at cell level
                        BS(n).num_adm_above_SAGBR(s)=BS(n).num_adm_above_SAGBR
                            (s)+1;
                    end;

                    if (config.aggregate_avg_Rb_multi_cell(s)+BS(n).Rbreq(s))>
                        config.SAGBR(s)
                        %Measure of SAGBR at the whole scenario
                        stats.num_adm_above_global_SAGBR(s)=stats.
                            num_adm_above_global_SAGBR(s)+1;
                    end;

                    changing_conditions=1;

                else
                    %Count a blocking:
                    BS(n).num_blocks(s)=BS(n).num_blocks(s)+1;
```

```matlab

                 %Compute statistics:
                if (BS(n).avg_bit_rate_assigned_per_tenant(s)+BS(n).Rbreq(s))<config.cellSAGBR(s)
                    %Measure of SAGBR at cell level
                    BS(n).num_rej_below_SAGBR(s)=BS(n).num_rej_below_SAGBR(s)+1;
                end;

                if (config.aggregate_avg_Rb_multi_cell(s)+BS(n).Rbreq(s))<config.SAGBR(s)
                    %Measure of SAGBR at the whole scenario
                    stats.num_rej_below_global_SAGBR(s)=stats.num_rej_below_global_SAGBR(s)+1;
                end;

            end;
            %Schedule the arrival of next session for the tenant s:

            BS(n).time_next_session_arrival(s)=BS(n).time_next_session_arrival(s)+(-1/BS(n).lambda(s))*log(1-rand());
            %NOTE: We schedule not after "time" but after
            %"time_next_session_arrival" allowing multiple calls in a
            %time step.

            %Register the arrival and the system state

            %LOG FORMAT
            %[Time,Tenant,Cell,Duration,Rbreq,Admit,Bitrate per tenant
            %(inst), Bitrate per tenant(avg), NumRBper tenant (inst),
            %NumRB per tenant (avg), DeltaC, DeltaCext, DeltaCbal, CongestionStatus]
            num_entries=num_entries+1;
            arrival_log(num_entries,1:6)=[time,s,n,duration_session,BS(n).Rbreq(s),admit];
            index_log=7;
            for naux=1:config.num_cells
                arrival_log(num_entries,index_log:(index_log+config.num_tenants-1))=BS(n).bit_rate_assigned_per_tenant;
                arrival_log(num_entries,(index_log+config.num_tenants):(index_log+2*config.num_tenants-1))=BS(n).avg_bit_rate_assigned_per_tenant;
                arrival_log(num_entries,(index_log+2*config.num_tenants):(index_log+3*config.num_tenants-1))=BS(n).num_assigned_RB_per_tenant;
                arrival_log(num_entries,(index_log+3*config.num_tenants):(index_log+4*config.num_tenants-1))=BS(n).avg_num_RB_per_tenant;
                arrival_log(num_entries,(index_log+4*config.num_tenants):(index_log+5*config.num_tenants-1))=BS(n).DeltaC;
                arrival_log(num_entries,(index_log+5*config.num_tenants):(index_log+6*config.num_tenants-1))=BS(n).DeltaCext;
                arrival_log(num_entries,(index_log+6*config.num_tenants):(index_log+7*config.num_tenants-1))=BS(n).DeltaCbal;
                arrival_log(num_entries,(index_log+7*config.num_tenants):(index_log+7*config.num_tenants))=BS(n).congestion_status;
                index_log=index_log+7*config.num_tenants+1;
            end;
        end;
    end;
end;
```

```

    %Assess performance for the current time step.
    %ONLY IF SOME CHANGE HAS OCCURRED (NEW UEs OR UES ending).
    if changing_conditions
        compute_occupation(config,BS);
    end;

    %1) Update the averages of
    %BS(n).avg_num_RB_per_tenant   %Average number of RBs used by each tenant.
    %BS(n).avg_RB_utilisation_per_tenant    %Average real capacity share (RB_utilisation) per tenant.
    %BS(n).Rb_estimate_per_RB              %Estimate of bit rate per RB (in kb/s) achieved in the cell
    %BS(n).avg_bit_rate_assigned_per_tenant

    for n=1:config.num_cells
        BS(n).total_assigned_RB_sample(t_index)=sum(BS(n).num_assigned_RB_per_tenant);

        for s=1:config.num_tenants
            BS(n).num_assigned_RB_per_tenant_sample(t_index,s)=BS(n).num_assigned_RB_per_tenant(s);
            BS(n).bit_rate_assigned_per_tenant_sample(t_index,s)=BS(n).bit_rate_assigned_per_tenant(s);

            if t_index<=config.time_window_utilisation_averaging_samples
                BS(n).avg_num_RB_per_tenant(s)=mean(BS(n).num_assigned_RB_per_tenant_sample(1:t_index,s));
                BS(n).avg_bit_rate_assigned_per_tenant(s)=mean(BS(n).bit_rate_assigned_per_tenant_sample(1:t_index,s));
            else
                BS(n).avg_num_RB_per_tenant(s)=mean(BS(n).num_assigned_RB_per_tenant_sample((t_index-config.time_window_utilisation_averaging_samples):t_index,s));
                BS(n).avg_bit_rate_assigned_per_tenant(s)=mean(BS(n).bit_rate_assigned_per_tenant_sample((t_index-config.time_window_utilisation_averaging_samples):t_index,s));
            end;

            BS(n).avg_num_RB_per_tenant_sample(t_index,s)=BS(n).avg_num_RB_per_tenant(s);
            BS(n).avg_RB_utilisation_per_tenant(s)=BS(n).avg_num_RB_per_tenant(s)/BS(n).num_RBs;
            BS(n).avg_RB_utilisation_per_tenant_sample(t_index,s)=BS(n).avg_RB_utilisation_per_tenant(s);

            BS(n).avg_bit_rate_assigned_per_tenant_sample(t_index,s)=BS(n).avg_bit_rate_assigned_per_tenant(s);

            %Compute data volume
            for i=1:BS(n).numUEs(s)
                BS(n).data_volume_per_tenant(s)=BS(n).data_volume_per_tenant(s)+BS(n).UElist{s}(i).S*BS(n).UElist{s}(i).assigned_RBs*config.B_RB*config.time_step;
            end;
        end;

        %Estimate of bit rate per RB:
        BS(n).total_assigned_bit_rate_sample(t_index)=sum(BS(n).bit_rate_assigned_per_tenant);
        %BS(n).bit_rate_assigned_per_tenant_sample(t_index,:)=BS(n).bit_rate_assigned_per_tenant(:);

        if t_index<=config.time_window_bit_rate_averaging_samples
            aggregated_Rb=sum(BS(n).total_assigned_bit_rate_sample(1:t_index));
            aggregated_RBs=sum(BS(n).total_assigned_RB_sample(1:t_index));
        else
            aggregated_Rb=sum(BS(n).total_assigned_bit_rate_sample((t_index-config.time_window_bit_rate_averaging_samples):t_index));
            aggregated_RBs=sum(BS(n).total_assigned_RB_sample((t_index-config.time_window_bit_rate_averaging_samples):t_index));
        end;

        if aggregated_Rb>0
            %Update the new average of Bit rate
            BS(n).Rb_estimate_per_RB=aggregated_Rb/aggregated_RBs;
        end;
        %Note: if aggregated_Rb=0, meaning that no bit rate has been
        %obtained in the last period, the value of Rb_estimate_per_RB is
        %kept to the initial value.
        BS(n).Rb_estimate_per_RB_sample(t_index)=BS(n).Rb_estimate_per_RB;

        %Check congestion status:
        if BS(n).congestion_status
            BS(n).num_congested_samples=BS(n).num_congested_samples+1;
        end;
    end;

    %2) Compute the average RB utilisation of each tenant at multi-cell level
    %and the aggregate bit rate of each tenant at multi-cell level
    for s=1:config.num_tenants
        config.C_avg_multi_cell(s)=0;
        config.aggregate_avg_Rb_multi_cell(s)=0;
        for n=1:config.num_cells
            config.C_avg_multi_cell(s)=config.C_avg_multi_cell(s)+BS(n).avg_RB_utilisation_per_tenant(s);
            config.aggregate_avg_Rb_multi_cell(s)=config.aggregate_avg_Rb_multi_cell(s)+BS(n).avg_bit_rate_assigned_per_tenant(s);
        end;
        config.C_avg_multi_cell(s)=config.C_avg_multi_cell(s)/config.num_cells;
        config.C_avg_multi_cell_samples(t_index,s)=config.C_avg_multi_cell(s);
    end;

    %3) Compute and update the delta_C parameters
    %Computation of deltaCext:
    for n=1:config.num_cells
        for s=1:config.num_tenants
            BS(n).DeltaCext_sample(t_index,s)=0;
            for saux=1:config.num_tenants
                if  saux~=s
                    %BS(n).DeltaCext_sample(t_index,s)=BS(n).DeltaCext_sample(t_index,s)+config.C(saux)-BS(n).avg_RB_utilisation_per_tenant(saux);
                    BS(n).DeltaCext_sample(t_index,s)=BS(n).DeltaCext_sample(t_index,s)+config.C(saux)*config.correction_factor-BS(n).avg_RB_utilisation_per_tenant(saux);
                end;
            end;
            BS(n).DeltaCext_sample(t_index,s)=max(BS(n).DeltaCext_sample(t_index,s),0);

            %Average:
            if t_index<=config.time_window_delta_averaging_samples
                BS(n).DeltaCext(s)=mean(BS(n).DeltaCext_sample(1:t_index,s));
            else
                BS(n).DeltaCext(s)=mean(BS(n).DeltaCext_sample((t_index-config.time_window_delta_averaging_samples):t_index,s));
            end;

            BS(n).DeltaCext_avg_sample(t_index,s)=BS(n).DeltaCext(s);
        end;
    end;

    %Computation of deltaCbal and DeltaC total:
    for s=1:config.num_tenants
        for n=1:config.num_cells
            BS(n).DeltaCbal_sample(t_index,s)=(config.num_cells-1)*config.C(s);
            for naux=1:config.num_cells
                if naux~=n
                    BS(n).DeltaCbal_sample(t_index,s)=BS(n).DeltaCbal_sample(t_index,s)-BS(naux).avg_RB_utilisation_per_tenant(s);
                end;
            end;

            %Average:
            if t_index<=config.time_window_delta_averaging_samples
                BS(n).DeltaCbal(s)=mean(BS(n).DeltaCbal_sample(1:t_index,s));
            else
                BS(n).DeltaCbal(s)=mean(BS(n).DeltaCbal_sample((t_index-config.time_window_delta_averaging_samples):t_index,s));
            end;
            BS(n).DeltaCbal_avg_sample(t_index,s)=BS(n).DeltaCbal(s);

            %Computation of DeltaC total:

            if BS(n).DeltaCext_sample(t_index,s)>BS(n).Cextra_min
                BS(n).DeltaC_sample(t_index,s)=BS(n).beta*BS(n).DeltaCext_sample(t_index,s);
            else
                BS(n).DeltaC_sample(t_index,s)=BS(n).gamma*BS(n).DeltaCbal_sample(t_index,s);
            end;

            %Average:
            if t_index<=config.time_window_delta_averaging_samples
                BS(n).DeltaC(s)=mean(BS(n).DeltaC_sample(1:t_index,s));
            else
                BS(n).DeltaC(s)=mean(BS(n).DeltaC_sample((t_index-config.time_window_delta_averaging_samples):t_index,s));
            end;
            BS(n).DeltaC_avg_sample(t_index,s)=BS(n).DeltaC(s);

        end;
    end;
end;

% Measure final statistics and plots.

%Average along the whole simulation:
stats.avg_Cshare_multicell=mean(config.C_avg_multi_cell_samples);

stats.num_blocks_total=zeros(1,config.num_tenants);
stats.num_session_attempts_total=zeros(1,config.num_tenants);
stats.offered_load_total=zeros(1,config.num_tenants);
stats.congestion_prob_per_cell=zeros(config.num_cells,1);

stats.num_adm_above_SAGBR_total=zeros(1,config.num_tenants); %For the
    cellSAGBR
stats.num_rej_below_SAGBR_total=zeros(1,config.num_tenants); %For the
    cellSAGBR

for n=1:config.num_cells
    stats.avg_Cshare_per_cell(n,:)=mean(BS(n).
        avg_RB_utilisation_per_tenant_sample);
    stats.avg_Cbal(n,:)=mean(BS(n).DeltaCbal_avg_sample);
    stats.avg_Cext(n,:)=mean(BS(n).DeltaCext_avg_sample);
    stats.avg_Ctot(n,:)=mean(BS(n).DeltaC_avg_sample);
    stats.avg_bit_rate_per_tenant_per_cell(n,:)=mean(BS(n).
        bit_rate_assigned_per_tenant_sample);
    for s=1:config.num_tenants
        stats.blocking_prob_per_cell(n,s)=BS(n).num_blocks(s)/BS(n).
            num_session_attempts(s);
        stats.num_blocks_total(s)=stats.num_blocks_total(s)+BS(n).num_blocks(s
            );
        stats.num_session_attempts_total(s)=stats.num_session_attempts_total(s
            )+BS(n).num_session_attempts(s);
        stats.offered_load_total(s)=stats.offered_load_total(s)+BS(n).
            offered_load(s);

        stats.adm_prob_above_SAGBR_per_cell(n,s)=BS(n).num_adm_above_SAGBR(s)/
            BS(n).num_session_attempts(s);
        stats.rej_prob_below_SAGBR_per_cell(n,s)=BS(n).num_rej_below_SAGBR(s)/
            BS(n).num_session_attempts(s);

        stats.num_adm_above_SAGBR_total(s)=stats.num_adm_above_SAGBR_total(s)+
            BS(n).num_adm_above_SAGBR(s);
        stats.num_rej_below_SAGBR_total(s)=stats.num_rej_below_SAGBR_total(s)+
            BS(n).num_rej_below_SAGBR(s);

        stats.data_volume_per_cell(n,s)=BS(n).data_volume_per_tenant(s)/(8*1E6
            );  %Measured in GByte
    end;
    stats.avg_session_rate_per_cell(n,:)=BS(n).num_session_attempts/config.
        simulation_duration;
    stats.offered_load_per_cell(n,:)=BS(n).offered_load/config.
        simulation_duration; %Measured in kb/s
    stats.congestion_prob_per_cell(n)=BS(n).num_congested_samples/t_index;
    stats.avg_RB_ocupation(n)=mean(BS(n).total_assigned_RB_sample);
end;
stats.blocking_prob_total=stats.num_blocks_total./stats.
    num_session_attempts_total;
stats.avg_session_rate_total=stats.num_session_attempts_total/config.
    simulation_duration;
stats.offered_load_total=stats.offered_load_total/config.simulation_duration;
stats.avg_bit_rate_per_tenant_total=sum(stats.avg_bit_rate_per_tenant_per_cell
    ,1);
stats.data_volume_per_tenant=sum(stats.data_volume_per_cell,1);

stats.adm_prob_above_SAGBR_total=stats.num_adm_above_SAGBR_total./stats.
    num_session_attempts_total; %For the cellSAGBR
stats.rej_prob_below_SAGBR_total=stats.num_rej_below_SAGBR_total./stats.
    num_session_attempts_total; %For the cellSAGBR

stats.adm_prob_above_GLOBAL_SAGBR=stats.num_adm_above_global_SAGBR./stats.
    num_session_attempts_total; %For the global SAGBR
stats.rej_prob_below_GLOBAL_SAGBR=stats.num_rej_below_global_SAGBR./stats.
    num_session_attempts_total; %For the global SAGBR

%generate_plots(config, BS);
```

```

arrival_log=arrival_log(1:num_entries,1:index_log-1); %To reduce the size of
    the arrival_log

name_output_file=['sim2CellsEqual_T1_',num2str(loop_variable),'_T2_0.2
    _DeltaCorrVAR.mat'];
save(name_output_file);

end;
```

Q-Learning algorithm

```
close all
clear all

%% Fuzzy Q-Learning %%

gamma = 0.7;     % discount factor
eta = 0.5;     % parameter learning factor
epsilon = 0.9;  % exploration probability (1-epsilon = exploit / epsilon =
    explore)

% states
for i=1:81
    state(i) = i;
end
% actions
action = [0.05,0,-0.05];
% initial Q matrix
q1 = zeros(length(state),length(action));
epoch = 500;      % maximum number of iterations
state_idx = zeros(epoch,1);
action_idx = zeros(epoch,1);
state_idx(1) = 15; % the initial state to begin from
ST = [];
FUZZY = [];
alpha_i = [];
Q1 = zeros(length(state),length(action));
matrix = zeros(epoch,3);
V_t = zeros(length(state),1);
act = zeros(epoch,1);
reward = zeros(epoch+1,1);
r1 = zeros(epoch,1);
r2 = zeros(epoch,1);
Pblock1 = zeros(epoch+1,1);
Pblock2 = zeros(epoch+1,1);
Pblock1_nd = zeros(epoch+1,1);
Pblock2_nd = zeros(epoch+1,1);
Pblock11 = zeros(epoch+1,1);
Pblock12 = zeros(epoch+1,1);
Pblock21 = zeros(epoch+1,1);
Pblock22 = zeros(epoch+1,1);
Pblock11_nd = zeros(epoch+1,1);
Pblock12_nd = zeros(epoch+1,1);
Pblock21_nd = zeros(epoch+1,1);
Pblock22_nd = zeros(epoch+1,1);
BitRate1 = zeros(epoch,1);
BitRate2 = zeros(epoch,1);
Q_error1 = zeros(epoch,1);
constant1 = -247;
constant2 = 50;
constant3 = 3;
delta = zeros(epoch,1);
```

```
delta_T1_C1=0;
delta_T1_C2=0;
delta_T2_C1=0;
delta_T2_C2=0;

fismat_T1_C2 = readfis('Fuzzy2-Train');
fismat_T2_C1 = readfis('Fuzzy3-Train');
fismat_T2_C2 = readfis('Fuzzy4-Train');

a = 0.05;
b = 0.3;
c = 0.75;
d = 1.15;
lambda_t1_c1 = (b-a).*rand(epoch,1) + a;
lambda_t1_c2 = (b-a).*rand(epoch,1) + a;
lambda_t2_c1 = (d-c).*rand(epoch,1) + c;
lambda_t2_c2 = (d-c).*rand(epoch,1) + c;

trimf_1 = [-0.2 0.2 0.6];
trapmf_1 = [-1.2 -0.4 -0.2 0.2];
trapmf_2 =  [0.2 0.6 0.8 1.6];

mean11 = 10;
mean12 = 30;
mean13 = 50;
mean21 = 10;
mean22 = 30;
mean23 = 50;
mean31 = 10;
mean32 = 30;
mean33 = 50;
mean41 = 10;
mean42 = 30;
mean43 = 50;
sigma11 = 6;
sigma12 = 11;
sigma13 = 6;
sigma21 = 6;
sigma22 = 11;
sigma23 = 6;
sigma31 = 6;
sigma32 = 11;
sigma33 = 6;
sigma41 = 6;
sigma42 = 11;
sigma43 = 6;

%% the main loop of the algorithm

for k = 1:epoch

            disp(['iteration: ' num2str(k)]);

            [OL_T1_C1,OL_T1_C2,OL_T2_C1,OL_T2_C2, Pblock_1, Pblock_2]=
                sim_AC_v3(lambda_t1_c1(k),lambda_t1_c2(k),lambda_t2_c1(k),
                lambda_t2_c2(k));
            OL = [OL_T1_C1,OL_T1_C2,OL_T2_C1,OL_T2_C2];
            OL_Cell_1 = [OL(1),OL(3)];
            OL_Cell_2 = [OL(2),OL(4)];
            Pblock = [Pblock_1, Pblock_2];

            ST(:,1) = exp(-(OL_Cell_1(1)-mean11)^2/(2*sigma11^2));      % Low
                OL1
```

```
        ST(:,2) = exp(-(OL_Cell_1(1)-mean12)^2/(2*sigma12^2));      %
            Medium OL1
        ST(:,3) = exp(-(OL_Cell_1(1)-mean13)^2/(2*sigma13^2));      % High
            OL1

        ST(:,4) = exp(-(OL_Cell_1(2)-mean21)^2/(2*sigma21^2));      % Low
            OL2
        ST(:,5) = exp(-(OL_Cell_1(2)-mean22)^2/(2*sigma22^2));      %
            Medium OL2
        ST(:,6) = exp(-(OL_Cell_1(2)-mean23)^2/(2*sigma23^2));      % High
            OL2

        ST(:,7) = trapmf(delta_T1_C1,trapmf_1);      % Low Delta_T1_C1
        ST(:,8) = trimf(delta_T1_C1,trimf_1);      % Medium Delta_T1_C1
        ST(:,9) = trapmf(delta_T1_C1,trapmf_2);     % High Delta_T1_C1

        ST(:,10) = trapmf(delta_T2_C1,trapmf_1);      % Low Delta_T2_C1
        ST(:,11) = trimf(delta_T2_C1,trimf_1);      % Medium Delta_T2_C1
        ST(:,12) = trapmf(delta_T2_C1,trapmf_2);     % High Delta_T2_C1

        %% Start Fuzzy variables %%
        FUZZY(:,1) = [ST(:,1) ST(:,4) ST(:,7) ST(:,10)];  % LLLL
        FUZZY(:,2) = [ST(:,1) ST(:,4) ST(:,7) ST(:,11)];  % LLLM
        FUZZY(:,3) = [ST(:,1) ST(:,4) ST(:,7) ST(:,12)];  % LLLH

        FUZZY(:,4) = [ST(:,1) ST(:,4) ST(:,8) ST(:,10)];  % LLML
        FUZZY(:,5) = [ST(:,1) ST(:,4) ST(:,8) ST(:,11)];  % LLMM
        FUZZY(:,6) = [ST(:,1) ST(:,4) ST(:,8) ST(:,12)];  % LLMH

        FUZZY(:,7) = [ST(:,1) ST(:,4) ST(:,9) ST(:,10)];   % LLHL
        FUZZY(:,8) = [ST(:,1) ST(:,4) ST(:,9) ST(:,11)];  % LLHM
        FUZZY(:,9) = [ST(:,1) ST(:,4) ST(:,9) ST(:,12)];  % LLH

        FUZZY(:,10) = [ST(:,1) ST(:,5) ST(:,7) ST(:,10)];  % LMLL
        FUZZY(:,11) = [ST(:,1) ST(:,5) ST(:,7) ST(:,11)];  % LMLM
        FUZZY(:,12) = [ST(:,1) ST(:,5) ST(:,7) ST(:,12)];  % LMLH

        FUZZY(:,13) = [ST(:,1) ST(:,5) ST(:,8) ST(:,10)];  % LMML
        FUZZY(:,14) = [ST(:,1) ST(:,5) ST(:,8) ST(:,11)];  % LMMM
        FUZZY(:,15) = [ST(:,1) ST(:,5) ST(:,8) ST(:,12)];  % LMMH

        FUZZY(:,16) = [ST(:,1) ST(:,5) ST(:,9) ST(:,10)];  % LMHL
        FUZZY(:,17) = [ST(:,1) ST(:,5) ST(:,9) ST(:,11)];  % LMHM
        FUZZY(:,18) = [ST(:,1) ST(:,5) ST(:,9) ST(:,12)];  % LMHH

        FUZZY(:,19) = [ST(:,1) ST(:,6) ST(:,7) ST(:,10)];  % LHLL
        FUZZY(:,20) = [ST(:,1) ST(:,6) ST(:,7) ST(:,11)];  % LHLM
        FUZZY(:,21) = [ST(:,1) ST(:,6) ST(:,7) ST(:,12)];  % LHLH

        FUZZY(:,22) = [ST(:,1) ST(:,6) ST(:,8) ST(:,10)];  % LHML
        FUZZY(:,23) = [ST(:,1) ST(:,6) ST(:,8) ST(:,11)];  % LHMM
        FUZZY(:,24) = [ST(:,1) ST(:,6) ST(:,8) ST(:,12)];  % LHMH

        FUZZY(:,25) = [ST(:,1) ST(:,6) ST(:,9) ST(:,10)];  % LHHL
        FUZZY(:,26) = [ST(:,1) ST(:,6) ST(:,9) ST(:,11)];  % LHHM
        FUZZY(:,27) = [ST(:,1) ST(:,6) ST(:,9) ST(:,12)];  % LHHH

        FUZZY(:,28) = [ST(:,2) ST(:,4) ST(:,7) ST(:,10)];  % MLLL
        FUZZY(:,29) = [ST(:,2) ST(:,4) ST(:,7) ST(:,11)];  % MLLM
        FUZZY(:,30) = [ST(:,2) ST(:,4) ST(:,7) ST(:,12)];  % MLLH

        FUZZY(:,31) = [ST(:,2) ST(:,4) ST(:,8) ST(:,10)];  % MLML
        FUZZY(:,32) = [ST(:,2) ST(:,4) ST(:,8) ST(:,11)];  % MLMM
```

```
            FUZZY(:,33) = [ST(:,2) ST(:,4) ST(:,8) ST(:,12)];  % MLMH

            FUZZY(:,34) = [ST(:,2) ST(:,4) ST(:,9) ST(:,10)];  % MLHL
            FUZZY(:,35) = [ST(:,2) ST(:,4) ST(:,9) ST(:,11)];  % MLHM
            FUZZY(:,36) = [ST(:,2) ST(:,4) ST(:,9) ST(:,12)];  % MLHH

            FUZZY(:,37) = [ST(:,2) ST(:,5) ST(:,7) ST(:,10)];  % MMLL
            FUZZY(:,38) = [ST(:,2) ST(:,5) ST(:,7) ST(:,11)];  % MMLM
            FUZZY(:,39) = [ST(:,2) ST(:,5) ST(:,7) ST(:,12)];  % MMLH

            FUZZY(:,40) = [ST(:,2) ST(:,5) ST(:,8) ST(:,10)];  % MMML
            FUZZY(:,41) = [ST(:,2) ST(:,5) ST(:,8) ST(:,11)];  % MMMM
            FUZZY(:,42) = [ST(:,2) ST(:,5) ST(:,8) ST(:,12)];  % MMMH

            FUZZY(:,43) = [ST(:,2) ST(:,5) ST(:,9) ST(:,10)];  % MMHL
            FUZZY(:,44) = [ST(:,2) ST(:,5) ST(:,9) ST(:,11)];  % MMHM
            FUZZY(:,45) = [ST(:,2) ST(:,5) ST(:,9) ST(:,12)];  % MMHH

            FUZZY(:,46) = [ST(:,2) ST(:,6) ST(:,7) ST(:,10)];  % MHLL
            FUZZY(:,47) = [ST(:,2) ST(:,6) ST(:,7) ST(:,11)];  % MHLM
            FUZZY(:,48) = [ST(:,2) ST(:,6) ST(:,7) ST(:,12)];  % MHLH

            FUZZY(:,49) = [ST(:,2) ST(:,6) ST(:,8) ST(:,10)];  % MHML
            FUZZY(:,50) = [ST(:,2) ST(:,6) ST(:,8) ST(:,11)];  % MHMM
            FUZZY(:,51) = [ST(:,2) ST(:,6) ST(:,8) ST(:,12)];  % MHMH

            FUZZY(:,52) = [ST(:,2) ST(:,6) ST(:,9) ST(:,10)];  % MHHL
            FUZZY(:,53) = [ST(:,2) ST(:,6) ST(:,9) ST(:,11)];  % MHHM
            FUZZY(:,54) = [ST(:,2) ST(:,6) ST(:,9) ST(:,12)];  % MHHH

            FUZZY(:,55) = [ST(:,3) ST(:,4) ST(:,7) ST(:,10)];   % HLLL
            FUZZY(:,56) = [ST(:,3) ST(:,4) ST(:,7) ST(:,11)];   % HLLM
            FUZZY(:,57) = [ST(:,3) ST(:,4) ST(:,7) ST(:,12)];   % HLLH

            FUZZY(:,58) = [ST(:,3) ST(:,4) ST(:,8) ST(:,10)];  % HLML
            FUZZY(:,59) = [ST(:,3) ST(:,4) ST(:,8) ST(:,11)];  % HLMM
            FUZZY(:,60) = [ST(:,3) ST(:,4) ST(:,8) ST(:,12)];  % HLMH

            FUZZY(:,61) = [ST(:,3) ST(:,4) ST(:,9) ST(:,10)];  % HLHL
            FUZZY(:,62) = [ST(:,3) ST(:,4) ST(:,9) ST(:,11)];  % HLHM
            FUZZY(:,63) = [ST(:,3) ST(:,4) ST(:,9) ST(:,12)];  % HLHH

            FUZZY(:,64) = [ST(:,3) ST(:,5) ST(:,7) ST(:,10)];  % HMLL
            FUZZY(:,65) = [ST(:,3) ST(:,5) ST(:,7) ST(:,11)];  % HMLM
            FUZZY(:,66) = [ST(:,3) ST(:,5) ST(:,7) ST(:,12)];  % HMLH

            FUZZY(:,67) = [ST(:,3) ST(:,5) ST(:,8) ST(:,10)];  % HMML
            FUZZY(:,68) = [ST(:,3) ST(:,5) ST(:,8) ST(:,11)];  % HMMM
            FUZZY(:,69) = [ST(:,3) ST(:,5) ST(:,8) ST(:,12)];  % HMMH

            FUZZY(:,70) = [ST(:,3) ST(:,5) ST(:,9) ST(:,10)];  % HMHL
            FUZZY(:,71) = [ST(:,3) ST(:,5) ST(:,9) ST(:,11)];  % HMHM
            FUZZY(:,72) = [ST(:,3) ST(:,5) ST(:,9) ST(:,12)];  % HMHH

            FUZZY(:,73) = [ST(:,3) ST(:,6) ST(:,7) ST(:,10)];  % HHLL
            FUZZY(:,74) = [ST(:,3) ST(:,6) ST(:,7) ST(:,11)];  % HHLM
            FUZZY(:,75) = [ST(:,3) ST(:,6) ST(:,7) ST(:,12)];  % HHLH

            FUZZY(:,76) = [ST(:,3) ST(:,6) ST(:,8) ST(:,10)];  % HHML
            FUZZY(:,77) = [ST(:,3) ST(:,6) ST(:,8) ST(:,11)];  % HHMM
            FUZZY(:,78) = [ST(:,3) ST(:,6) ST(:,8) ST(:,12)];  % HHMH

            FUZZY(:,79) = [ST(:,3) ST(:,6) ST(:,9) ST(:,10)];  % HHHL
```

```
        FUZZY(:,80) = [ST(:,3) ST(:,6) ST(:,9) ST(:,11)];  % HHHM
        FUZZY(:,81) = [ST(:,3) ST(:,6) ST(:,9) ST(:,12)];  % HHHH

        %% End Fuzzy variables %%

        for i=1:81
            alpha_i(k,i) = FUZZY(1,i)*FUZZY(2,i)*FUZZY(3,i)*FUZZY(4,i);
        end

        [strength_max,state_idx_max] = max(alpha_i(k,:));

        if k~=1
            state_idx(k) = state_idx_max;
            % Compute the value of the new state %
            for i=1:length(state)
                    V_t(state_idx(k)) = alpha_i(k,i)*max(q1(i,action_idx(k
                        -1))) + V_t(state_idx(k));
            end
            % Calculate the error signal %
            Q_error1(k) = reward(k) + gamma*V_t(state_idx(k))-Q1(state_idx
                (k-1),action_idx(k-1));

            % Update q-values by an ordinary gradient descent method %
            q1(state_idx(k),umax(i)) = q1(state_idx(k),umax(i)) + eta*
                Q_error1(k)*alpha_i(k-1,state_idx(k));
        end

        r=rand; % get 1 uniform random number
        prob_area=sum(r>=cumsum([0, 1-epsilon, epsilon])); % check it to
            be in which probability area

        for i=1:length(state)
            % choose either explore or exploit
            if prob_area == 1   % exploit
                    [~,umax(i)]=max(q1(i,:));
                    a_i(i) = action(umax(i));
            else        % explore
                [a_i(i),umax(i)] = datasample(action,1); % choose 1 action
                    randomly (uniform random distribution)
            end
        end

        % Calculate the global action %

        for i=1:length(state)
            act(k) = act(k) + alpha_i(k,i)*a_i(i);
        end

        [c index] = min(abs(action-act(k)));
        action_idx(k) = find(action==action(index)); % id of the chosen
            action

        % Approximate the Q-function from the current q-values and the
            degree
        % of truth of the rules

        for i=1:length(state)
                    Q1(state_idx(k),action_idx(k)) = alpha_i(k,i)*q1(i,
                        umax(i))+Q1(state_idx_max,action_idx(k));
        end

        for(i=1:81)
            q_values1(i,k) = Q1(i,1);
```

```matlab
                q_values2(i,k) = Q1(i,2);
                q_values3(i,k) = Q1(i,3);
                q_values4(i,k) = Q1(i,4);
                q_values5(i,k) = Q1(i,5);
            end

            %% Evolve to the next state %%

            % Observe the reinforcement signal %

            delta_T1_C1 = delta_T1_C1 + action(action_idx(k));

            delta_T1_C2 = evalfis(OL,fismat_T1_C2);
            delta_T2_C1 = evalfis(OL,fismat_T2_C1);
            delta_T2_C2 = evalfis(OL,fismat_T2_C2);

            [Pblock1(k+1),Pblock2(k+1),Pblock11(k+1),Pblock12(k+1),Pblock21(k
                +1),Pblock22(k+1)]=sim_AC_v4(lambda_t1_c1(k),lambda_t1_c2(k),
                lambda_t2_c1(k),lambda_t2_c2(k),delta_T1_C1,delta_T1_C2,
                delta_T2_C1,delta_T2_C2);
            [Pblock1_nd(k+1),Pblock2_nd(k+1),Pblock11_nd(k+1),Pblock12_nd(k+1)
                ,Pblock21_nd(k+1),Pblock22_nd(k+1)]=sim_AC_v6(lambda_t1_c1(k),
                lambda_t1_c2(k),lambda_t2_c1(k),lambda_t2_c2(k),delta_T1_C1,
                delta_T1_C2,delta_T2_C1,delta_T2_C2);

            r1(k) = log10((1+1/((Pblock1(k+1)+0.1)*1000)));
            r2(k) = log10((1+1/((Pblock2(k+1)+0.1)*1000)));

            reward(k+1) = 100*(r1(k)+r2(k))+0.1357;

            % conditions
            if (delta_T1_C1 <= -0.3 || delta_T1_C1 >= 0.65)
                delta_T1_C1 = 0.2;
            end

          % Update epsilon
          epsilon = epsilon - (1/650);
end
```

Bibliography

[1] P. Cerwall, "Ericsson mobility report," *MWC*, 2016.

[2] A. Imran, A. Zoha, and A. Abu-Dayya, "Challenges in 5G: how to empower SON with big data for enabling 5G," *IEEE Network*, vol. 28, no. 6, pp. 27 – 33, 2014. DOI: 10.1109/MNET.2014.6963801.

[3] COMARCH, "Towards Self-Organizing Networks," *White Paper*, 2009.

[4] M. Dohler, "5G Ultra-High Capacity Network Design With Rates 10x LTE-A," *IEEE ComSoc Distinguished Lectureship Tour Texas/Arizona USA*, 2012.

[5] S. Yang, D. Frangopol, and L. Neves, "Service life prediction of structural systems using lifetime functions with emphasis on bridges," *Reliability Engineering & System Safety*, vol. 86, no. 1, pp. 39–51, 2004.

[6] R. Razavi, S. Klein, and H. Claussen, "A Fuzzy reinforcement learning approach for self-optimization of coverage in LTE networks," *Bell Labs Technical Journal*, vol. 15, no. 3, pp. 153 – 175, 2010. DOI: 10.1002/bltj.20463.

[7] X. Wang, X. Li, and V. Lueng, "Artificial Intelligence-Based Techniques for Emerging Heterogeneous Network: State of the Arts, Opportunities, and Challenges," *IEEE Access*, vol. 3, pp. 1379 – 1391, 2015. DOI: 10.1109/ACCESS.2015.2467174.

[8] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2014. ISBN: 978-0-262-01243-0.

[9] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[10] C. Jiang, H. Zhang, Y. Ren, Z. Han, K.-C. Chen, and L. Hanzo, "Machine Learning Paradigms for Next-Generation Wireless Networks," *IEEE Wireless Communications*, vol. 24, pp. 98 – 105, 2016. DOI: 10.1109/MWC.2016.1500356WC.

[11] C. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[12] J. Bennett, "Machine Learning, part III: The Q-learning algorithm." `https://articles.wearepop.com/secret-formula-for-self-learning-computers`, 2016.

[13] Q. Li, H. Xia, Z. Zeng, and T. Zhang, "Dynamic Enhanced Inter-Cell Interference Coordination using Reinforcement Learning Approach in Heterogeneous Network," *Proceedings of ICCT2013*, 2013. DOI: 10.1109/ICCT.2013.6820379.

[14] M. Bennis, S. Perlaza, P. Blasco, Z. Han, and V. Poor, "Self-Organization in Small Cell Networks: A Reinforcement Learning Approach," *IEEE Transactions on Wireless Communications*, vol. 12, no. 7, pp. 3202 – 3212, 2013. DOI: 10.1109/TWC.2013.060513.120959.

[15] Z. Zhenzhen, C. Jie, and N. Crespi, "A Policy-Based Framework for Autonomic Reconfiguration Management in Heterogeneous Networks," *Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia*, pp. 71–78, 2008. DOI: 10.1145/1543137.1543150.

[16] S. Fan, H. Tian, and C. Sengul, "Self-optimization of coverage and capacity based on a fuzzy neural network with cooperative reinforcement learning," *EURASIP Journal on Wireless Communications and Networking*, 2014. DOI: 10.1186/1687-1499-2014-57.

[17] X. Yang, S. Chien, and T. Ting, *Bio-Inspired Computation in Telecommunications.* Morgan Kaufmann, 2015. ISBN: 978-0-12-801538-4.

[18] Y. Song, L. Liu, H. Ma, and A. Vasilakos, "A Biology-Based Algorithm to Minimal Exposure Problem of Wireless Sensor Networks," *IEEE Transactions on Network and Service Management*, vol. 11, no. 3, pp. 417 – 430, 2014. DOI: 10.1109/TNSM.2014.2346080.

[19] N. Hasan, W. Ejaz, N. Ejaz, H. Kim, A. Anpalagan, and M. Jo, "Network Selection and Channel Allocation for Spectrum Sharing in 5G Heterogeneous Networks," *IEEE Access*, vol. 4, pp. 980 – 992, 2016. DOI: 10.1109/ACCESS.2016.2533394.

[20] J. Reddy and N. Kumar, "Computational algorithms inspired by biological processes and evolution," *Current science*, vol. 103, no. 4, pp. 370–380, 2012.

[21] O. Abdelkhalek, S. Krichen, A. Guitouni, and S. Mitrovic-Minic, "A genetic algorithm for a multi-objective nodes placement problem in heterogeneous network infrastructure for surveillance applications," *Wireless and Mobile Networking Conference (WMNC), 2011 4th Joint IFIP*, 2011. DOI: 10.1109/WMNC.2011.6097214.

[22] W. Mai, H.-L. Liu, L. Chen, J. Li, and H. Xiao, "Multi-objective Evolutionary Algorithm for 4G Base Station Planning," *Ninth International Conference on Computational Intelligence and Security*, pp. 85–89, 2013.

[23] X. Liang, H. Liu, and Q. Wang, "4G Heterogeneous Networks Base Station Planning Using Evolutionary Multi-objective Algorithm," *11th International Conference on Computational Intelligence and Security (CIS)*, 2015. DOI: 10.1109/CIS.2015.68.

[24] L. Ho, I. Ashraf, and H. Claussen, “Evolving femtocell coverage optimization algorithms using genetic programming,” *IEEE 20th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2009. DOI: 10.1109/PIMRC.2009.5450062.

[25] M. Millonas, “Swarms, phase transitions, and collective intelligence,” *Artificial Life III*, pp. 417–445, 1993.

[26] Z. Zhang, K. Long, J. Wang, and F. Dressler, “On Swarm Intelligence Inspired Self-Organized Networking: Its Bionic Mechanisms, Designing Principles and Optimization Approaches,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 513–537, 2013. DOI: 10.1109/SURV.2013.062613.00014.

[27] H. Shokrani and S. Jabbehdari, “A Survey of Ant-Based Routing Algorithms for Mobile Ad-hoc Networks,” *International Conference on Signal Processing Systems*, 2009. DOI: 10.1109/ICSPS.2009.29.

[28] L. Wenjing, Y. Peng, J. Zhengxin, and L. Zifan, “Centralized Management Mechanism for Cell Outage Compensation in LTE Networks,” *International Journal of Distributed Sensor Networks*, vol. 8, no. 11, pp. 1–8, 2012.

[29] R. Han, C. Feng, H. Xia, and Y. Wu, “Coverage Optimization for Dense Deployment Small Cell Based on Ant Colony Algorithm,” *Vehicular Technology Conference (VTC Fall)*, 2014. DOI: 10.1109/VTCFall.2014.6965924.

[30] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Prentice Hall PTR, 1998. ISBN: 0132733501.

[31] R. Chai, J. Cheng, X. Pu, and Q. Chen, “Neural Network Based Vertical Handoff Performance Enhancement in Heterogeneous Wireless Networks,” *7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, 2011. DOI: 10.1109/wicom.2011.6036665.

[32] A. Calhan and C. Ceken, “An adaptive neuro-fuzzy based vertical handoff decision algorithm for wireless heterogeneous networks,” *IEEE 21st International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, 2010. DOI: 10.1109/PIMRC.2010.5671693.

[33] Z. Han and R. Liu, *Resource Allocation for Wireless Networks: Basics, Techniques, and Applications.* Cambridge University Press, 2008. ISBN: 0521873851.

[34] MATLAB & Simulink - MathWorks, “Foundations of fuzzy logic.” `https://es.mathworks.com/help/fuzzy/foundations-of-fuzzy-logic.html`, 2016.

[35] K. Vasudeva, S. Dikmese, I. Guven, A. Mehbodniya, W. Saad, and F. Adachi, “Fuzzy-Based Game Theoretic Mobility Management for Energy Efficient Operation in HetNets,” *IEEE Access*, vol. 5, pp. 7542–7552, 2017. DOI: 10.1109/ACCESS.2017.2689061.

[36] B. Ma and X. Liao, "Speed-adaptive vertical handoff algorithm based on fuzzy logic in vehicular heterogeneous networks," *9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2012. DOI: 10.1109/FSKD.2012.6234358.

[37] A. Daeinabi, K. Sandrasegaran, and P. Ghosal, "An enhanced intercell interference coordination scheme using fuzzy logic controller in LTE-advanced heterogeneous networks," *International Symposium on Wireless Personal Multimedia Communications (WPMC)*, 2014. DOI: 10.1109/WPMC.2014.7014873.

[38] Small Cell Forum, "Market drivers for multi-operator small cells," *Document SCF 017.06.01*, 2016.

[39] 3GPP TS 23.251 v13.1.0, "Network Sharing; Architecture amd functional description (Release 13)," 2015.

[40] J. Pérez-Romero, O. Sallent, R. Ferrús, and R. Agustí, "Admission control for multi-tenant Radio Access Networks," *IEEE International Conference on Communications Workshops (ICC Workshops)*, 2017. DOI: 10.1109/ICCW.2017.7962801.

[41] J. Pérez-Romero, O. Sallent, R. Ferrús, and R. Agustí, "Self-optimsed admission control for multi-tenant Radio Access Networks," *Draft.*

[42] 3GPP TS 36.300 v13.2.0, "E-UTRA and E-UTRAN Overall description; Stage 2 (Release 13)," 2015.

[43] P. Munoz, R. Barco, I. Bandera, M. Toril, and S. Luna-Ramirez, "Optimization of a Fuzzy Logic Controller for Handover-Based Load Balancing," *IEEE 73rd Vehicular Technology Conference (VTC Spring)*, 2011. DOI: 10.1109/VETECS.2011.5956148.

[44] J. Pérez-Romero, O. Sallent, R. Ferrús, and R. Agustí, "Knowledge-based 5G Radio Access Network planning and optimization," *International Symposium on Wireless Communication Systems (ISWCS)*, 2016. DOI: 10.1109/ISWCS.2016.7600929.

[45] "View on 5G Architecture," *5G PPP Architecture Working Group*, 2014.