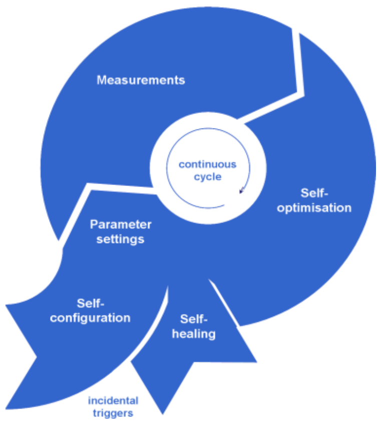
Master of Science Thesis

Self-Optimized Resource Allocation in ICT Systems



Behnaz Shirmohamadi

Network Architectures and Services (NAS) Group
Department of Telecommunications
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Performance of Networks and Systems (PoNS) Group
Technical Sciences Expertise Center
TNO

Self-Optimized Resource Allocation in ICT Systems

Master of Science Thesis

For the degree of Master of Science in
Network Services and Architectures Group (NAS)
at Department of Telecommunications
at Delft University of Technology

by

Behnaz Shirmohamadi

September 26, 2011

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft, The Netherlands

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF
TELECOMMUNICATIONS

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled

Self-Optimized Resource Allocation in ICT Systems

by

Behnaz Shirmohamadi

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE.

Dated: September 26, 2011

Supervisors:

Prof. Dr. Ir. Robert E. Kooij
Dr.Ir. Remco Litjens
Prof. Dr .Hans van den Berg

Readers: Dr. Ir. Gerard Janssen

Abstract

In this thesis, I have investigated the self-optimization approach in order to solve a generic resource allocation challenge. The challenge is defined for a general ICT system serving two classes of jobs: low and high priority. The high priority jobs require a higher quality of service compared to the low priority jobs.
In order to fulfill this difference in QoS levels, a part of the total resource capacity should be held reserved to serve only high priority jobs. Two different self-optimization methods are applied to solve the challenge and the objective of the self-optimization algorithms is to split the total resources in such a way as to minimize the overall Blocking considering the different level of QoS. The first applied method is a rule-based method and the other one is fuzzy-Q learning.
I also have defined a performance quality matrix which is used to assess and compare the algorithm's reactions in three sets of designed simulation scenarios. The first set of scenarios aims to examine the effect of the different parameter settings on the overall performance of the algorithms. The second set of scenarios simulates a partial failure in the total capacity which is subsequently repaired causing the system to return to normalcy, and observes the algorithm's reactions in adapting to these changes. The final set of simulations changes the arrival process to batch arrival where in one group of simulations the arrival rate (λ) has not changed while in the other it has decreased by the rate of the average arrival batch size.

Key Words: self-optimization, resource allocation, rule-based, fuzzy-Q learning.

Acknowledgements

First and foremost, I would like to thank my supervisors Dr. ir. Remco Litjens and Prof. dr. Hans van den Berg, from the Dutch organization for applied scientific research (TNO) for their excellent guidance and valuable comments on simulations and designing scenarios. I highly appreciate their contributions and constant supervisions. Moreover, I would like to thank my supervisor from Delft University of Technology (TU Delft), Prof. dr. ir. Robert E. Kooij for his support during this thesis. I also wish to thank all my colleagues from TNO ICT for their support and encouragement throughout the thesis work.
I am so grateful towards my aunt and uncle who never let me feel far from my family. Their warm hearts have saved me from freezing in cold long winters of The Netherlands. I am also very thankful to my dear friends, Yeganeh, Nadjla, Nazgol and Laleh for being kind and supportive during my study.
I know that I can never express my appreciation to my parents as much as I do feel it deeply inside but I would like to thank my dearest Leila and Masoud for their unconditional love and support. Whatever I have accomplished in my life is because of them standing by me all the time.
And my last but not least special thanks go to Wilfred who could “de zon laten schijnen” whenever my life was dark.

Delft, The Netherlands Behnaz Shirmohamadi
September 26, 2011

Dedicated to people who have reinforced me in learning

Table of Contents

List of Figures

List of Tables

1 Introduction

In this chapter, I will first briefly introduce the concept of self-organizing and self-optimizing systems, focusing primarily on optimal resource utilization. I will then outline the scope of this research, and, finally, provide an overview of the structure of content and chapters of this work.

1.1 General Introduction

Over the course of the last few decades, ICT systems have become significantly more complex than their predecessors. Today's ICT systems are larger, have more users, consist of many novel elements and must manage a diversity of inter and intra system interactions. Although the changes that ICT systems have undergone have greatly improved these systems in a number of ways, they have also made it difficult and time consuming for operators to set and manage these complex systems manually. In order to decrease the complexity of manually setting and managing these systems while minimizing capital expenditures (CAPEX) and operational expenditures (OPEX) as well as enhancing the performance quality of systems, self-organizing systems happen to be of great interest to operators [1].

Researchers generally divide self-organizing algorithms into three different subareas: self-configuration, self-healing and self-optimization [2]. Each subarea of these algorithms attempts to enhance a self-organized system in a particular way. For each subarea, there are automated methodologies which aim to improve the performance and quality of service (QoS) by reacting to the dynamic processes related to the system. First, self-configuration methodologies deal with the integration of newly deployed services or features and the reconfiguration of the entire system by adapting to the changes in topology [3]. Second, self-healing mechanisms attempt to reduce the impact of the potential failure of the system by enabling the system to be recovery oriented in a way that it can response to the failure appropriately and return the system to a state of normalcy [4]. Finally, self-optimization algorithms automatically and continuously tune and adapt system settings and parameters according to the dynamic variation of the system and characteristics of the environment without (or with limited) human intervention. The self-optimizing subarea is the focus of the thesis.

A self-optimizing algorithm repeatedly adjusts the system's parameters in order to adapt to a variety of changes that the system may experience. For example, changes can happen in the traffic profile, system characteristics or interaction between the system and other systems or between the system and its environment. A self-optimization algorithm is a closed loop process of parameter deployment, performance and quality evaluation, parameter optimization, and redeployment of newly optimized parameters to the system. This closed loop functions to improve the system's performance constantly, and, consequently guarantee a certain level of quality of the performance in the system.

Different scholars have studied self-optimization algorithms' applications in various fields of ICT systems and networks, such as wireless mobile networks and cloud computing. One of the most interesting instances of employing self-optimization methods in wireless mobile networks is related to the settings of handover parameters which allows a network to provide a higher QoS and more efficient inter and intra cell resource allocation [5]. Self-optimizing methods are also used in the field of wireless mobile network power management in order to use power resources more optimized [6]. Additionally, operators have implemented self-organizing methods with cloud computing, specifically in the realms of green cloud computing [7] and also resource management parameters [8] are well known target areas for implementing self-organizing methods. The aforementioned uses of these algorithms indicate that resource utilization can be a vital field of interest for self-optimizing systems because the resources are mostly limited and expensive, which forces cost-conscious stakeholders to use them in an efficient way. Therefore, implementing self-organization and self-optimization methods is a way to use limited resources to offset the cost of manual setting and the complexity of the work.

1.2 Scope of the Project

Understanding the importance of optimal resource utilization and the role of self-organizing algorithms in achieving more efficient and better resource allocation in ICT systems, this project identifies two resource allocation challenges in two different ICT system domains. One of the challenges is resource splitting in a cloud computing server serving two classes of customers (first and second class), while the other considers the challenge of resource allocation between fresh and handover calls in a cellular mobile network system. Both of these challenges are presented as one general case study which can be applied to any system of generic resource allocation between two classes of jobs requiring two different levels of QoS. The case study that I will present is an ICT system with a given capacity (resource) to serve two different classes of jobs: high priority and low priority. The challenge is splitting the given fixed resource into two categories of resources: shared capacity and reserved capacity for high priority jobs.

Accordingly, I have applied two different self-optimization algorithms to achieve the most optimal resource split in relation to the self-optimization algorithms' objective which is to minimize the overall weighted Blocking probabilities. One of the applied self-optimization methods is a rule-based method, which works based on if-then rules written by a human expert. The other one is the fuzzy-Q learning algorithm. This learning algorithm does not have any a-priori knowledge about the system and proper decision making logic, but it does have the ability to learn from former experiences and to take appropriate actions in different system states.

This thesis focuses on observing, assessing and comparing the performance of the mentioned self-optimization algorithms in the following three sets of simulation scenarios:

- Different parameter settings
- A sudden failure in the capacity
- Change in arrival process

The first set of simulation scenarios investigates the effect of parameter settings on the overall performance of the system. The second simulates an actual case of partial failure in the server (in cloud computing) or antenna (in cellular network) and, subsequently, determines the ability of algorithms to adapt to this failure. It also considers the case of repairing the failure and examines the ability of self-optimization algorithms to adjust the system back to the normal working situation. The last simulation set models a shift in the arrival process from the Poisson process to the batch arrival of a Geometric distribution. This shift models a sudden change in the arrival profile due to for example introduction of a new service (in cloud computing) or a failure in the neighboring cell (in cellular mobile networks).

1.3 Structure of the Thesis

The first part of Chapter 2 describes two resource allocation challenges in cloud computing and cellular mobile network domains, while the rest of the chapter focuses on modeling these two resource allocation problems into a widely applicable case study. Chapter 3 first provides a general overview of two applied self-optimization methods and then describes the steps of implementation of each algorithm on the considered case study outlined in Chapter 2. Chapter 4 presents three sets of simulation scenarios and the results of each scenario as well as an in-depth analysis of the results and findings. The simulation results are used to assess the self-optimization algorithms' performance based on performance quality metric, especially the converged value of the cumulative Blocking, the convergence time of the cumulative Blocking and the time fraction of meeting targeted Blocking defined by the operator. Chapter 5 summarizes the work, draws conclusions and suggests directions for further research.

2 Case Study

In this chapter, I first provide a brief overview of cloud computing and cellular wireless mobile networks. I then identify a resource allocation challenge in each domain. While the resource allocation challenges that I discuss are different, they can be modeled into one general case study. Finally, I evaluate this case study, which is used in simulations, in the last part of this chapter.

2.1 An Introduction to Cloud Computing

The basic concept of cloud computing was introduced in the 1960s. At the beginning, it was a vague idea of sharing infinite computational resources in which a few remote locations provide computation and unimpeded access for global users. Although, this striking term was introduced several decades ago, distributed computing did not draw significant attention until the last decade. In the last few years, due to the enormous increase in the number of Internet users, services and new web-oriented devices, the need for distributed computational resources has risen dramatically. Eventually, the long-held dream of computing as a utility started to be realized in October 2007 when IBM and Google announced their collaboration in the cloud computing domain and soon after, IBM introduced its “Blue Cloud” project. Since then, “cloud computing” has become one of today’s most popular concepts in the computer world [9].
Today, cloud computing refers to the applications that are delivered as services over the Internet as well as the system hardware and software in the datacenters providing those services. The software and hardware of the datacenter is called a “cloud”. A cloud is referred to as a “public cloud” if it is available to the public and if the utility computing can be sold as a service in a pay-as-you-go offer [10].

2.1.1 Cloud Computing Case Study

Since one of the fundamental concepts of cloud computing is considering the computation as a service, I have identified a resource allocation challenge in a public cloud server as follows. Assume a public cloud computing system with a given total resource capacity of C_{Total} offering two price classes of computational services: first and second class. A customer asking for a higher level of QoS pays a first class price which is higher than the second class price. The second class customer naturally receives a lower quality of service. Within this system, customers are rejected if there is not enough vacant resources available to serve the customer and logically, rejecting a first class customer is penalized more strongly than rejecting a second class customer. To indicate this relative importance, a term of $\beta > 1$ is introduced as: β = the penalty of rejecting a high class customer / the penalty of rejecting a low class customer and β value is corresponding to the price difference. Accordingly, C_{Total} should be split into two parts: C_{Shared} to serve both first and second class customers and $C_{Reserved}$ which is reserved capacity meant only for serving first class customers.

Customers’ requests are generated according to a Poisson process, and each job request occupies one unit (channel) of the resource. A job requested by a first class customer is accepted if $N_{First} + N_{Second} < C_{Total}$ (N_{First} *and* N_{Second} represent the number of existing first and second class customers’ jobs using the resources), otherwise the request is rejected. In other words, a request from a first class customer is accepted if there is any vacant resource channel in the whole capacity ($C_{Total} = C_{Shared} + C_{Reserved}$). A job requested by a second class customer is admitted only if $N_{First} + N_{Second} < C_{Shared}$. So, a second class customer's job is accepted if any unit of resource in the shared capacity (C_{Shared}) is vacant.

The Blocking probabilities of first and second class job requests are observed and measured during an observation time and are reported to the decision maker agent of the system - also called the controller agent - as the system state (s(t)).

$s(t) = (P_{Second}(t) , P_{First}(t)),$

where

P_{First} = Blocked requests from first class customers / received requests from first class customers.

P_{Second} = Blocked requests from second class customers / received requests from second class customers jobs.

The goal of the self-optimization algorithm is to split C_{Total} into C_{Shared} and $C_{Reserved}$ in such a way as to minimize the overall Blocking *(B)* where $B = P_{Second} + \beta P_{First}$.

2.2 Wireless Mobile Network Introduction

A cellular mobile network is a mobile network in which the area under coverage is divided into subareas called cells. Each cell is served by at least one antenna located in a control point called the "base station". These cells, when they are connected, provide coverage over a wide geographic area. This enables a large number of mobile devices to communicate with each other and other fixed transceivers anywhere within a large area in the network. Mobility is one of the most important features of such a network, while continuous service and connectivity for the mobile terminal can be provided by supporting handover from one cell to another. Handover is the process of changing the former channel associated with the former base station located in an former cell to a channel associated with the current base station within a current cell while a call (or another service) is in progress [11]. This process is what allows us to easily keep talking with our mobile phones when driving from one city to another one without experiencing any inconvenience or disconnection while crossing many cells borders.

2.2.1 Wireless Mobile Network Case Study

According to the aforementioned principal characteristics of a wireless mobile network due to mobility and handover mentioned above, each base station within a cell serves two kinds of calls: fresh calls and handover calls. Fresh calls are those that originate in the cell and handover calls are calls that originate in other cells and have been transferred to the cell because the user has moved to a new cell while on the phone. Assume a total capacity of C_{Total} is dedicated to each cell and one call request requires one channel (unit) of the C_{Total} for establishing a connection. A call is accepted if any vacant channel is available in the cell otherwise it is blocked. Blocking a handover call is penalized more than blocking a fresh call because blocking a handover call means interruption and disconnection in the middle of an ongoing call which is irritating for users. The relative importance of a handover call over a fresh call is indicated by factor $\beta > 1$.

To avoid the user irritation by blocking a handover call, each cell reserves some channels to only serve handover calls to reduce the handover blocking probabilities. Therefore C_{Total} is split into $C_{Reserved}$ and C_{Shared} ($C_{Total} = C_{Reserved} + C_{Shared}$). C_{Shared} serves both fresh and handover calls while $C_{Reserved}$ only serves handover calls. Consequently, a fresh call is accepted if $N_{Fresh} + N_{Handover} < C_{Shared}$ (N_{Fresh} and $N_{Handover}$ are the present fresh and handover calls using the cell resources), otherwise it is blocked. Conversely, a handover call is accepted if $N_{Fresh} + N_{Handover} < C_{Total}$ or in other words, a handover call is accepted if any channel is vacant in the whole of the capacity of the cell.

Blocking probabilities of fresh and handover calls are measured in observation intervals and are reported to the decision maker agent of the system called the controller agent in order to provide information about the cell state (S(t)).

$S(t) = \; S(t) = (P_{Fresh}\,(t) \, , P_{Handover}(t)) \, ,$

where

$P_{Handover}$ = Blocked handover calls / Received handover calls

P_{Fresh} = Blocked fresh calls / Received fresh calls

The controller agent's goal is to split C_{Total} into $C_{Reserved}$ and C_{Shared} in order to achieve an overall minimized $P_{Fresh} + \beta P_{Handover}$.

2.3 Considered Case Study

The considered case study should be able to model both of the cases described above. Therefore, the case study system I have introduced is a resource with 35 channels in total, serving two types of jobs: low and high priority jobs. Blocking high priority jobs is penalized more than blocking low priority jobs presented by the factor $\beta > 1$ (β is the relative importance of the high priority jobs over low priority jobs). In the defined case study $\beta = 10$; therefore, blocking a high priority job is penalized 10 times more strongly than blocking a low priority job. High priority jobs represent handover calls or first class customers which need to be treated with a higher level of importance (demanding higher level of QoS) and lower blocking probabilities. Low priority jobs are normal jobs which do not require special treatment like fresh calls or second class jobs. Jobs are generated according to the Poisson process, where λ (expected number of occurrences per time unit) is 30. Of all the arrivals, 60% are high priority jobs and the rest are low priority jobs. Jobs have independent duration times, according to the Poisson distribution. The average duration is $1/\mu = 0.8$ seconds (for both classes of jobs).

All 35 channels of the system must be split into $C_{Reserved}$ and C_{Shared}. C_{Shared} is a common capacity serving both types of jobs and $C_{Reserved}$ signifies certain channels which are reserved for high priority jobs (Figure 2-1 and 2-2).

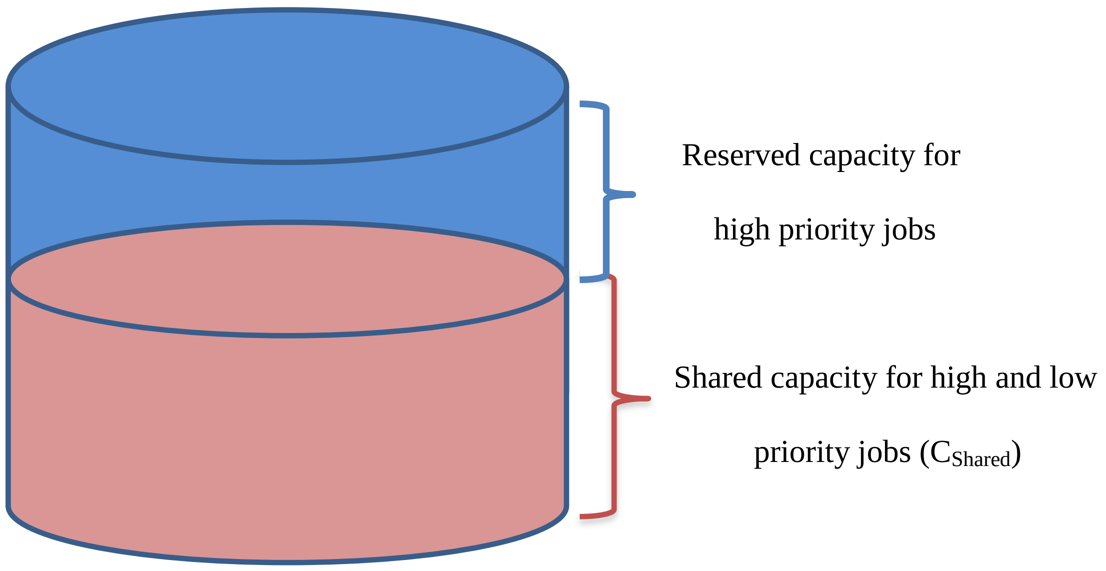

Reserved capacity for
high priority jobs
Shared capacity for high and low
priority jobs (C_{Shared})

Figure 2.1: A simple schematic presenting the considered case study system

The duration of the observation time is 100 seconds. In other words, each 100 seconds, blocking probabilities of high and low priority jobs are observed and measured as P_{High} and P_{Low}.

P_{High}= Blocked high priority jobs in an observation interval / Received high priority jobs in an observation interval.

P_{Low} = Blocked low priority jobs in an observation interval / Received low priority jobs in an observation interval.

Measuring blocking probabilities, the system state consisting of P_{Low} and P_{High} ($S(t) = (P_{Low}(t), P_{High}(t))$ is reported to the controller agent in order to provide the necessary information. The task of the controller agent is to act appropriately based on the reported state of the system and to update the C_{Shared} value. The self-optimizing algorithm's objective is to set and adjust the C_{Shared} value in a way to minimize the overall Blocking during the system operation.

Blocking $B(t) = P_{Low}(t) + 10P_{High}(t)$

The C_{Shared} value is initialized at 10 and the controller should modify C_{Shared} until it reaches the optimal value in relation to the self-optimization algorithms' objective

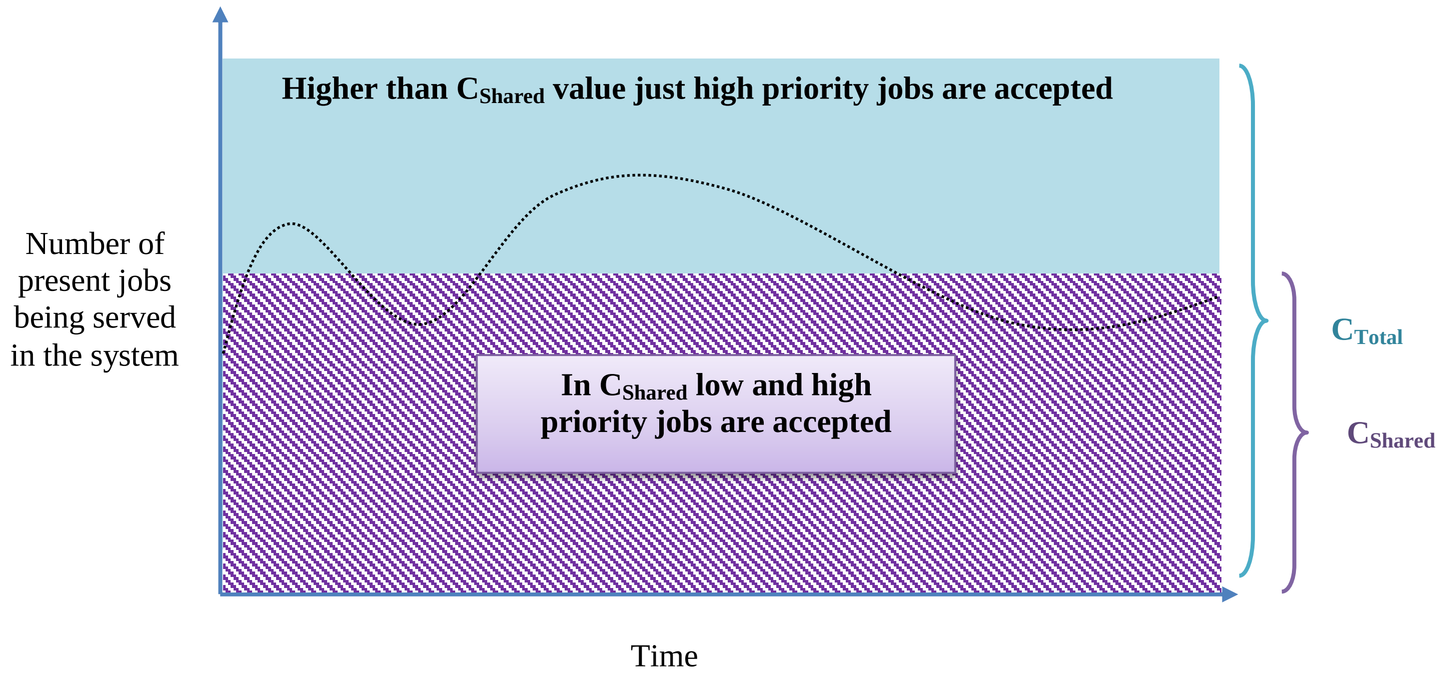

Higher than C_{Shared} value just high priority jobs are accepted
In C_{Shared} low and high priority jobs are accepted
Number of present jobs being served in the system
Time
C_{Total}
C_{Shared}

Figure 2.2: Job distribution diagram in C_{shared} and $C_{Reserved}$

3 Self-Optimization Methodologies

In this Chapter, first I give a general introduction about the conceived self-optimization methodologies which are applied to the case study. The rest of the chapter is dedicated to describing the specific application of these methods to the case study defined and designed in the Chapter 2.

3.1 Rule-Based Method

Rule-based systems or expert systems are somehow the first and simplest realization of the research in the field of Artificial Intelligence (AI). Rule-based systems are a way of implementing human knowledge applicable in a specific automated system [12]. The controller agents mostly do not have the ability of learning from former experiences and improving the performance based on what they have learned. They also lack the ability to expand their expertise and cannot deal with a new situation if it is not defined for them by an expert. In other words, rule-based systems are the devices to convey expert's knowledge for solving the problem to the machine and translating that knowledge into an understandable language for an automated system.

The question arises whether or not there is always a need for human knowledge to be transformed in to the artificial intelligent system language. Why do not we use the same human expert as the controller agent? There are three main advantages of translating human knowledge to rule-based systems. The first advantage is that the human expert's knowledge becomes available to a larger range of people and applications. Another advantage is that in this way the knowledge and expertise of the human expert can be captured, saved and protected against being lost when they retire or leave the firm. The last but not least advantage is that after once transferring the knowledge from expert to the rule-based system, the human involvement can be eliminated (or at least reduced) and the operating expense (OPEX) can be considerably decreased [13].

A typical rule-based system consists of at least three components [14]:

- The system interface
- The knowledge data base
- The controller agent

The system interface is responsible for measuring the quality of performance in the system and interprets this information in a data format which is comprehensible for the rule-based controller agent. For example the system interface measures the packet loss in a web server during a time interval and reports this as the system state to the controller agent. The controller agent is the brain of a rule-based system. It maps the received system state to the knowledge data base and derives the appropriate action according to the look-up table written by an expert. The knowledge data base is the expert knowledge required for decision making and problem solving translated to the form of if-then commands. These if-then commands are the core element of rule-based decision making. Some examples are given below:

- If X is Green $\longrightarrow$ Decrease the power.
- If Y = 1000 $\longrightarrow$ Turn 90 degree to the right.
- If Z decreases $\longrightarrow$ Set the temperature to 60º C.

The knowledge data base is a storage memory full of if-then rules covering all the possible system states. After receiving a new system state, the controller agent refers to this stack of rules and derives the suitable decision and action from this data base.
A schematic diagram of rule-based system units and connections is showed in Figure 3-1.

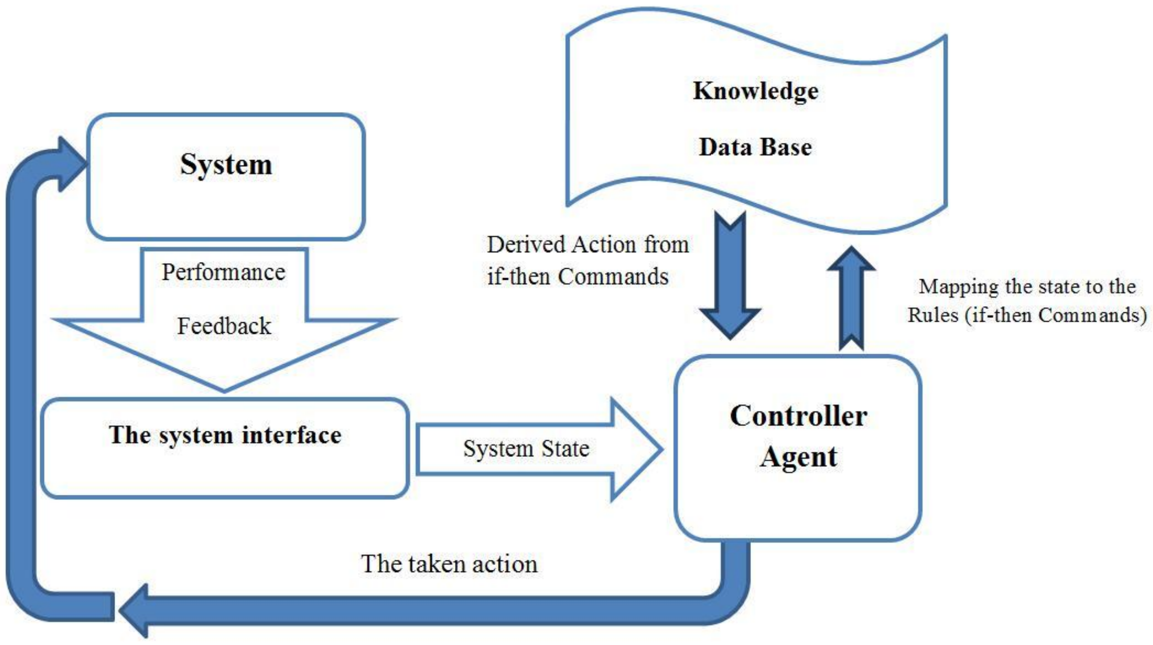


Figure 3.1: A simple diagram of a generic rule-based algorithm

3.2 Reinforcement Learning

The first inspiration for reinforcement learning (RL) arose from the nature. In the animal and human being nature it is the sign of intelligence to act in ways that are rewarded. As a human being we learn how to interact with our environment and what we have learnt about the environment and its reward and punishment policies become one of the main factors that help us in decision making procedures later on [15]. In the standard framework of reinforcement learning, a controller agent interacts with an unknown environment and tries to maximize a long-term benefit. A learning controller agent repeatedly observes the state of the system and then chooses and performs an action. Each executed action changes the system state and the agent also receives an immediate payoff as a result of the taken action. Positive payoffs can be considered as rewards and negative payoffs are punishments. The agent, exploiting the knowledge it has previously obtained through past actions and received rewards or punishments, must learn to take actions so as to maximize a long term sum or average payoff it receives in the future [16]. For example a chess player observes the game and the opponent pieces and makes a move planning and anticipating possible replies. One or some steps further it would be clear whether the player's move was a good choice or not. If it turned out as a good move in that circumstance, it is more likely for the player to repeat it in the future games when the game state and condition is the same.

Reinforcement learning is very different than supervised methodologies like rule-based systems. In the rule-based method, the needed knowledge for making decisions is provided by a knowledgeable external supervisor but in RL, the agent does not need a set of training or intelligence in advance; instead it learns on-line and can continuously learn and adopt while performing the required task [17]. This interactive approach is more suitable for problems for which deriving desired behavior and action in all the possible situations is not easy or practical. In this kind of problems, even a knowledgeable expert cannot predict and choose a best possible action for each state because unknown elements can affect the feedback rewards and punishments. With reinforcement learning the environment is considered unknown and unpredictable with dynamic parameters and characteristics.

Another important advantage of interactive methods like RL over supervised methods like rule-based is the opportunity of using exploration phase and exploitation phase simultaneously. To achieve the highest possible reward, the agent must choose an action which has been selected before and found to be successful to bring rewards to the system, but to discover such an action, the agent has to try actions which are not taken before in order to discover whether those actions are more appropriate or not. Therefore, the action not only must exploit what is has already learnt from former experiments but also is has to explore for the best possible decision in the future [18]. The main challenge of the controller agent is to find a healthy balance between exploration and exploitation phases without the failure in its basic task as the decision maker of the system [19].

It should be considered that exploration phase (occasionally choosing a random action rather than the action with highest quality) inherently involves potentially costly experimentation as an investment towards future possible benefits.

Figure 3-2 illustrates the core components of reinforcement learning algorithms.

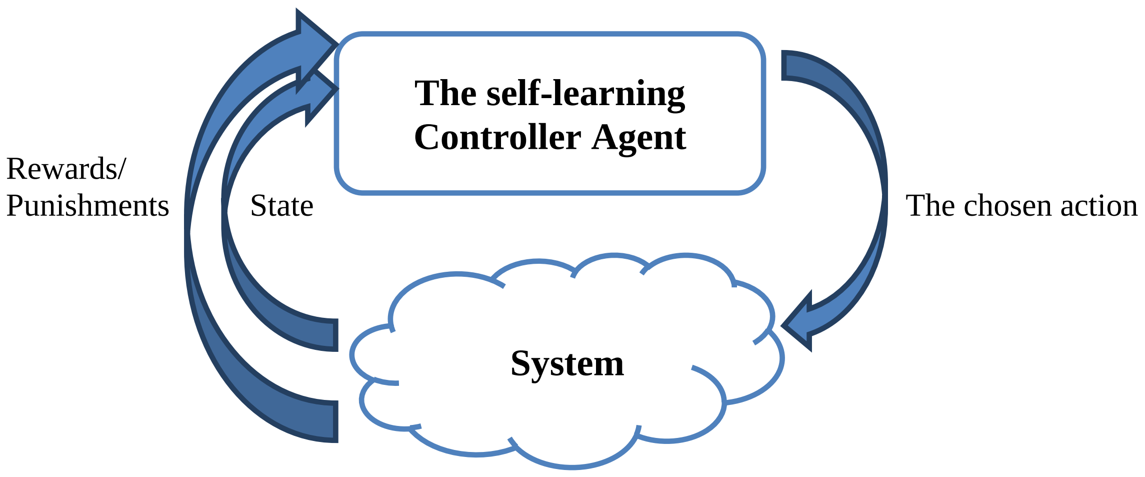


Figure 3.2: The components of a reinforcement learning algorithm

The controller agent is the learner and decision maker of the process, which interacts with the system via execution of actions and the reception of rewards (or punishments) depending on the taken action. The agent receives also information describing the environment state continuously. The reward or punishment achieved by a taken action as a feedback is sent back to the controller agent who uses this feedback for an assessment of the taken action. If the taken action brought reward to the system it is considered as a good action and it is likely to be chosen again later when the system is in the same state.

3.2.1 Markov Decision Process

Understanding the principles of the RL method's decision making process is not possible without introducing the Markov Decision Process (MDP). MDP provides a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker. MDP is an ideal mathematical way of modeling a self-optimized decision making process as the received feedback by the agent after taking an action can be effected by different environmental factors like fluctuation in the arrival process or system characteristics. More precisely, a Markov Decision Process is a discrete time stochastic control process containing four components [20]:

- A set of possible system states: S (a set of several s)
- A set of possible actions for each state: A (a set of several a)
- A real valued reward function: r(s(t),a(s(t)))
- A state transition function, which specifies probabilistically the next state of the environment, given its present state and agent's chosen action: $P_a(s(t),s(t+\Delta t))$.

At each time step, the system is in a state s(t). Considering s(t) the controller agent makes a decision and chooses action a(s(t)) from action set A which is available for the specific state s(t). The system responds to the taken action a(s(t)) at the next time step and moves to a new state s(t+Δt) and as a feedback sends to the agent a corresponding reward r (s(t),a(s(t))).

The core problem of MDPs is to find a *policy* for the decision maker (controller agent): a function π that specifies the action a = π(*s(t)*) that the decision maker chooses when the system is in the state s(t). The goal is to choose a policy *π* that will maximize the discounted cumulative reward over an infinite time where the discounted cumulative reward is:

$$\sum_{t=1}^{\infty} \gamma^t r_{a(t)}(s(t), s(t+\Delta t)),$$

where the chosen action based on the policy *π* is *a(t)*in which *π(s(t))=a(t)* and γ is the discount factor which satisfies 0 < γ < 1. γ <1 is needed to keep the cumulative reward finite and it is also used as a measure that indicates the relative importance of future rewards over instantaneous reward. γ is typically close to 1, which means the controller agent assigns equal value to immediate reward and future reward.

Given the state transition function *P* and the reward function *r*, the objective is to determine the policy that maximizes the expected discounted reward. To calculate this optimal policy, two arrays indexed by state *s(t)* needs to be stored: *value V*, which contains expected future values, and *policy π* which contains actions. At the end of the algorithm, *π* will contain the solutions and *V(s(t))* will contain the discounted averaged sum of the rewards to be earned by following that solution from state *s(t)*. The optimal action can be found a follows [21]:

$$a = \pi\,(s(t)) := \arg\max_a \{ \sum_{s(t+\Delta t)} P_a(s(t), s(t+\Delta t)(r(s(t), a(s(t))) + \gamma V(s(t+\Delta t))) \}$$

where

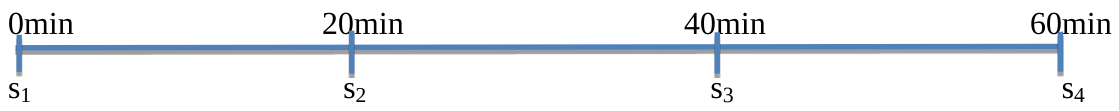
$$V(s(t)) := \sum_{s(t+\Delta t)} P_a(s(t), s(t+\Delta t))(r(s(t), a(s(t))) + \gamma V(s(t+\Delta t)))$$

3.2.2 Fuzzy-Q Learning

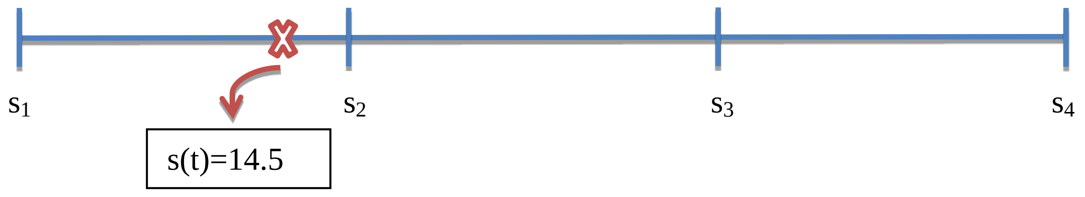
One of the reinforcement learning algorithms which attempt to find the optimal policy using MDP principles is Q-learning. The second conceived self-optimization method is Fuzzy-Q Learning which combines the fuzzy logic algorithm for discretizing the continuous state (s(t)) variable to ensure a finite number of sub-states (s_1, s_2, etc.) that forms the set of state(S), and Q-learning method as its decision making and learning algorithm.

3.2.2.1 Fuzzy Logic

In a fuzzy-Q learning algorithm, the reported continuous valued state (s(t)) is mapped to one or more sub-states of the S [22]. For example consider that s(t) is the continuous time a customer waits to be served at a server and assume that the maximum possible waiting time is one hour. Let the set S contain four sub-states denoted as (s_1, s_2, s_3 and s_4). These four sub-states divide the whole range of one hour as below:



The average waiting time is reported to the controller agent as 14.5 minutes (s(t)=14.5). This average waiting time should be mapped to one or more sub-states. The location of s(t) in the time diagram is between s_1 and s_2.



s(t) should be mapped to these two sub-states. As it is obvious in the diagram, s(t) is much more closer to s_2 than to s_1 and s(t) should be mapped to s_1 and s_2 in a weighted way to make it clear that even though s(t) is member of both s_1 and s_2 but is rather closer to s_1 than s_2. One way of implementing this weighted mapping is using the inverse distance. In this method the weight of membership of s(t) to a sub-state has reversed relation to the distance of s(t) to that sub-state. This weighted membership is called membership degree. Using this method s(t) = 14.5 can be mapped to s_1 and s_2 as below:

- s(t) distance to s_1 = 14.5 – 0 = 14.5 min
- s(t) distance to s_2 = 20 – 14.5 = 5.5 min
- distance between s_1 and s_2 is = 20 min

- membership degree of s(t) to $s_1 = \frac{5.5\ min}{20\ min} = 0.275$
- membership degree of s(t) to $s_2 = \frac{14.5\ min}{20\ min} = 0.725$

So s(t) is mapped to s_1 with the weight of 0.275 and is mapped to s_2 with the weight of 0.725.

3.2.2.2 Q-learning

Q-learning [23] is a well-known reinforcement learning technique which is more suitable to apply to systems where the reward values and probabilities of state transition function are not a priory known. The learning process of the Q-learning algorithm is entirely based on former experiences. Like other RL algorithms, the Q-learning method considers a set of states S and a set of actions per state A. By performing an action ($a \in A$) the state moves from the state s(t) to a new state s(t+1). This transition provides the agent an immediate reward value and the goal of agent is to maximize the long-term discounted cumulative reward. The Q-learning method tries to maximize this long-term discounted cumulative reward by learning which action is optimal for each sub-state based on former experiences.

The algorithm therefore has a function which calculates the quality of a (state,action) combination. These Q values indicate how good one action in a specific state is in relation to the optimization objective.

$Q: S \times A \longrightarrow \mathbf{R}$ (set of the real numbers)

Initially, all the Q values for all the possible (state,action) pairs are fixed to zero (initialized value). Subsequently each time the controller agent picks an action a(t) in state s(t) and will update the quality value of that specific (state,action) pair Q values based on the feedback rewards it has received from the system. The core of the algorithm is a simple value iteration update. It assumes the old Q(s(t),a(s(t))) values and makes a correction based on the newly received rewards, as follows [24]:

$$\underbrace{Q(s(t),a(s(t)))}_{\text{New value}} \leftarrow \underbrace{Q(s(t),a(s(t)))}_{\text{Old value}} + \alpha \times [\overbrace{\underbrace{r(s(t))}_{\text{Reward}} + \underset{\overset{\downarrow}{\text{Discount factor}}}{\gamma}\ \underbrace{\max_{a(s(t+\Delta t))} Q(s(t+\Delta t), a(s(t+\Delta t))}_{\text{Max future value}}}^{\text{Learned value}} - \underbrace{Q(s(t),a(s(t)))}_{\text{Old value}}]$$

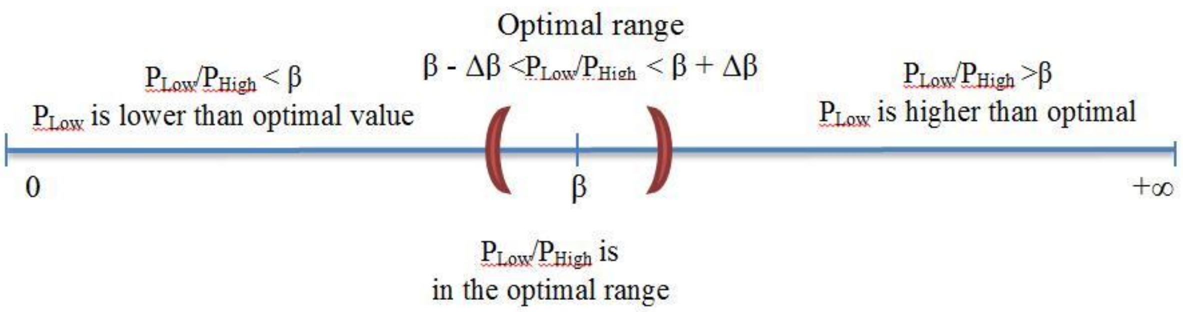
Here, r(s(t)) is the feedback reward to the controller agent after performing a(s(t)) in the state s(t), α ($0 < \alpha \leq 1$) is the learning rate and γ is discount factor such that $0 \leq \gamma < 1$.
The learning rate (α) determines to what extent the newly acquired information affects the old quality values. The taken action a(t) is judged by two terms: long-term performance and short-term performance. The short-term performance of a(t) is evaluated by immediate reward r(t) and the long-term performance is assessed by the difference between immediate quality and discounted future quality. If $\alpha = 0$, the agent does not learn anything from the experiences, while for a learning factor set to 1, the agent applies the newly received information (long-term and short-term action's performance assessment) to a full extent which means the recently received feedback affects the Q values strongly.
The discount factor γ indicates the importance of the immediate rewards over future rewards. A discount factor of 0 will make the agent "opportunistic" by only considering immediate rewards, while a discount factor close to 1 assigns equal values on immediate reward and future rewards [25].

3.3 Applied Self-Optimization Algorithms

In the following section, I describe the steps of applying the conceived methods (rule-based method and fuzzy-Q learning method) to the case study defined in Chapter 2.

3.3.1 Applying a Rule-Based Method to the Considered Case Study

Recalling the case study described in Chapter 2, the mission of the controller agent is using the rule-based algorithm as the self-optimization method in order to set C_{Shared} to the most optimal value in order to minimize the Blocking ($B(t) = P_{Low}(t) + \beta P_{High}(t)$) or maximize the reward ($r(t) = - P_{Low}(t) - \beta P_{High}(t)$). The reward function is a negative value and maximizing reward in this case means attempting to achieve $r(t) = 0$, i.e. no blocking.
Note that the self-optimization algorithms objective is maximizing the reward $r(t) = - P_{Low}(t) - \beta P_{High}(t)$ during the system performance. However this objective is translated in a different way for each method because of inherent differences of the methodologies. For the rule-based method, the objective is achieving the highest immediate reward in each observation priod, while the fuzzy-Q learning method's objective is maximizing the average discounted cumulative reward.
An observation agent observes the blocking probability of high and low priority jobs during the observation time Δt and reports these probabilities as the system state to the controller agent: $s(t) = (P_{Low}(t), P_{High}(t))$. Given $\beta = 10$ (the relative importance of high priority jobs over low priority jobs), the controller agent knows that blocking of high priority jobs is 10 times more strongly penalized than blocking of low priority jobs. Using this insight, the controller attempts to minimize $P_{Low} + \beta P_{High}$ by balancing $P_{Low}/P_{High} = \beta$. Therefore the whole range of P_{Low}/P_{High} is divided into three ranges as following


Optimal range
β - Δβ <PLow/PHigh < β + Δβ
PLow/PHigh < β
PLow is lower than optimal value
PLow/PHigh >β
PLow is higher than optimal
0
β
+∞
PLow/PHigh is
in the optimal range

These three range categories for P_{Low} and P_{High} are the main bases for establishing rules (If-then sentences). Three if-then rules are designed to provide the necessary information for the controller agent in order to make an appropriate decision and updating the C_{Shared} value. Given $\beta = 10$ and $\Delta\beta = 0.5$, the designed rules for the controller agent are as following:

- **If** $P_{Low} / P_{High} > 10.5$ **Then** low priority jobs experience high blocking probability $\longrightarrow$ increase C_{Shared} by one channel
- **If** $P_{Low} / P_{High} < 9.5$ **Then** high priority jobs experience high blocking probability $\longrightarrow$ decrease C_{Shared} by one channel
- **If** $9.5 < P_{Low} / P_{High} < 10.5$ **Then** $\longrightarrow$ do not change the C_{Shared} value

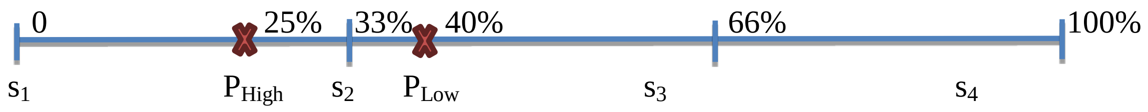
The controller agent maps the received state $s(t) = (P_{Low}(t),P_{High}(t))$ to one of the three rules and will derive the appropriate action. The chosen action will be implemented to the system. The observation agent resets blocking probabilities to zero and again starts observing and calculating the new records of blocked jobs in order to compute the new $s(t+\Delta t)$ to report to the controller agent at the end of next observation time interval $(t+\Delta t)$.

3.3.2 Applying a Fuzzy-Q Learning Method to the Considered Case Study

In this part the fuzzy-Q Learning is implemented on the designed case study in Chapter 2. The task of the fuzzy-Q learning algorithm is to help the controller agent in the decision making process for setting and updating C_{Shared} to the most optimal value in relation to the optimization objective which is maximizing the long-term discounted cumulative reward.

The state of the system is reported to the controller agent as $s(t) = (P_{Low}(t),P_{High}(t))$. $P_{Low}(t)$ and $P_{High}(t)$ are continuous blocking probability of low and high priority jobs and need to be discretized in order to be mapped in to the corresponding sub-states. The discretization process is done according to the fuzzification method outlined in Section 3-2-2-1.

For example assume in one observation time interval (Δt) P_{Low} and P_{High} are measured and reported to the agent as $s(t) = (P_{Low}(t),P_{High}(t)) = (0.4,0.25)$. The state set (S) for both P_{Low} and P_{High} consists of four sub-states denoted as (s_1, s_2, s_3 and s_4).



The fuzzification method outlined in Section 3-2-2-1 is used to derive membership degrees of P_{Low} and P_{High}:

$$P_{Low}(t) = 0.4 \longrightarrow \left.\begin{array}{l} \frac{40-33}{66-33} \approx 0.2 \\ \frac{66-40}{66-33} \approx 0.8 \end{array}\right\} \begin{array}{l} \text{Membership degree of } P_{Low} \text{ to sub-state } s_3 = 0.2 \\ \text{Membership degree of } P_{Low} \text{ to sub-state } s_2 = 0.8 \end{array}$$

$$P_{High}(t) = 0.25 \longrightarrow \left.\begin{array}{l} \frac{25-0}{33-0} \approx 0.76 \\ \frac{33-25}{33-0} \approx 0.24 \end{array}\right\} \begin{array}{l} \text{Membership degree of } P_{High} \text{ to sub-state } s_2 = 0.24 \\ \text{Membership degree of } P_{High} \text{ to sub-state } s_1 = 0.76 \end{array}$$

For each job class i ∈ {low, high} the membership degree vector (μ) is defined as:
$\mu_i(s(t)) = (\mu_i^1(s(t),\mu_i^2(s(t),\mu_i^3(s(t),\mu_i^4(s(t))$
The membership degree vector for high and low priority jobs is calculated as below:
$\mu_{Low}(s(t)) = (0,0.8,0.2,0)$ and $\mu_{High}(t) = (0.24,0.76,0,0)$.

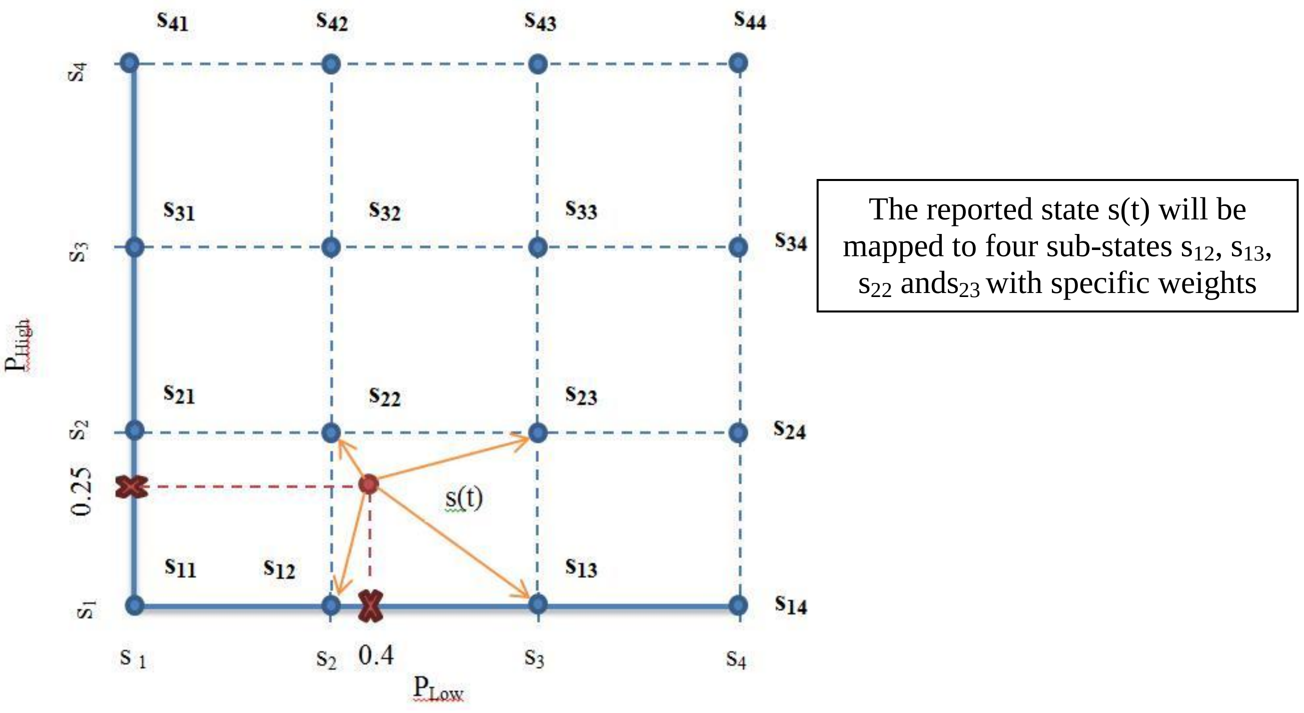


Figure 3.3: Fuzzified sub-states diagram

The overall state of the system can be descried by a two dimensional diagram illustrated in Figure 3.3:

Each bullet represents a sub-state and the reported state s(t) should be mapped to these sub-states weighted with membership degrees.
These class-specific membership degrees map blocking probabilities for each class separately, but what is needed is to map the whole state (combining high and low priority jobs membership degrees) to the whole sub-state set to provide a general fuzzified overview of the

system state considering both blocking probabilities. A 4×4 matrix is introduced to give the overall membership degree matrix $\boldsymbol{\mu}_{ij}$ (s(t)).

$\boldsymbol{\mu}_{ij}$ (s(t)) = (μ^{i}_{High} (s(t)) . μ^{j}_{Low} (s(t))

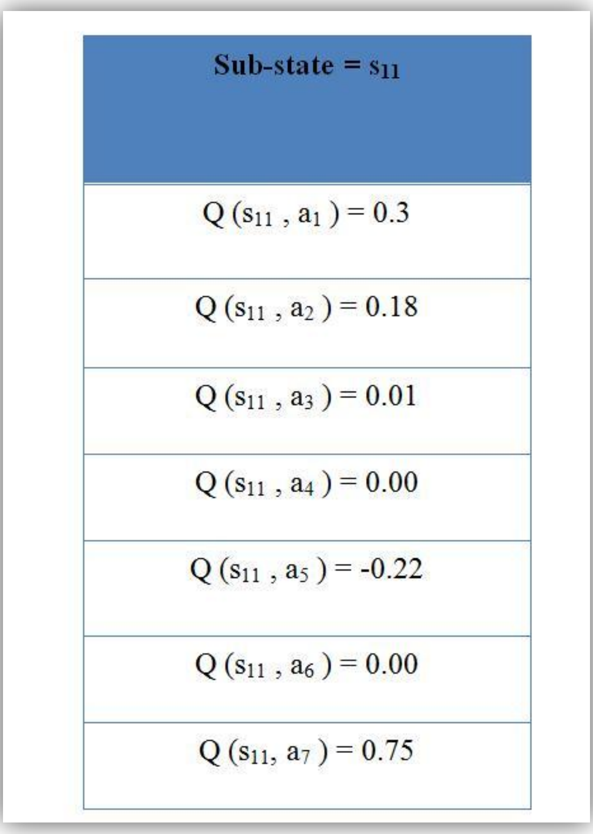
s_{11} $\mu_{11} = \mu^{1}_{High} . \mu^{1}_{Low}$ = 0	s_{12} $\mu_{12} = \mu^{1}_{High} . \mu^{2}_{Low}$ =0.19	s_{13} $\mu_{13} = \mu^{1}_{High} . \mu^{3}_{Low}$ = 0.05	s_{14} $\mu_{14} = \mu^{1}_{High} . \mu^{4}_{Low}$ =0
s_{21} $\mu_{21} = \mu^{2}_{High} . \mu^{1}_{Low}$ = 0	s_{22} $\mu_{22} = \mu^{2}_{High} . \mu^{2}_{Low}$ = 0.61	s_{23} $\mu_{23} = \mu^{2}_{High} . \mu^{3}_{Low}$ = 0.15	s_{24} $\mu_{24} = \mu^{2}_{High} . \mu^{4}_{Low}$ = 0
s_{31} $\mu_{31} = \mu^{3}_{High} . \mu^{1}_{Low}$ = 0	s_{32} $\mu_{32} = \mu^{3}_{High} . \mu^{2}_{Low}$ = 0	s_{33} $\mu_{33} = \mu^{3}_{High} . \mu^{3}_{Low}$ = 0	s_{34} $\mu_{34} = \mu^{3}_{High} . \mu^{4}_{Low}$ = 0
s_{41} $\mu_{41} = \mu^{4}_{High} . \mu^{1}_{Low}$ = 0	s_{42} $\mu_{42} = \mu^{4}_{High} . \mu^{2}_{Low}$ = 0	s_{43} $\mu_{43} = \mu^{4}_{High} . \mu^{3}_{Low}$ = 0	s_{44} $\mu_{44} = \mu^{4}_{High} . \mu^{4}_{Low}$ = 0

Table 3-1: A numerical example of the membership matrix

The membership matrix entries indicate how strongly s(t) is mapped to each sub-state. After mapping the state of the system to the discrete sub-states, the controller agent, should take a proper action to improve the state. The designed action set contains seven sub-actions as given below:

a_1 : increase C_{Shared} by 3 channels (units)	$\longrightarrow$	Corresponding active value +3
a_2 : increase C_{Shared} by 2 channels (units)	$\longrightarrow$	Corresponding active value +2
a_3 : increase C_{Shared} by 1 channels (units)	$\longrightarrow$	Corresponding active value +1
a_4 : decrease C_{Shared} by 1 channels (units)	$\longrightarrow$	Corresponding active value -1
a_5 : decrease C_{Shared} by 2 channels (units)	$\longrightarrow$	Corresponding active value -2
a_6 : decrease C_{Shared} by 3 channels (units)	$\longrightarrow$	Corresponding active value -3
a_7 : do not change the C_{Shared} value	$\longrightarrow$	Corresponding active value 0

To discover the best possible action, the controller agent refers to a look-up table contains the quality values (Q) of the all possible (sub-state,action) pairs. This look-up table helps the controller to find the best action for each sub-state according to former decisions and feedbacks. For example, assume for the sub-state s_{11}, the look-up table looks like the one below (Table 3-2):



Sub-state = s_{11}
$Q(s_{11}, a_1) = 0.3$
$Q(s_{11}, a_2) = 0.18$
$Q(s_{11}, a_3) = 0.01$
$Q(s_{11}, a_4) = 0.00$
$Q(s_{11}, a_5) = -0.22$
$Q(s_{11}, a_6) = 0.00$
$Q(s_{11}, a_7) = 0.75$

Table 3-2: A numerical example (sub-state, action) quality look-up table. The shown value are made up numbers just to illustrate the methodology

A similar table exists for each sub-state. Based on this look-up table information, which is updated based on former experiences, the best action for this sub-state is a_7, i.e. not to change the C_{Shared} value. After passing the time needed for convergence of Q values from initialized values, the quality values of (sub-state,action) pairs converge and the best action for each sub-state is known.

At the beginning, the algorithm is totally in the exploration phase because all the Q(sub-state,action) values are initialized to zeros and all the taken actions are random actions to explore the best action in each sub-state. As time passes the controller agent updates the quality values after taking any action and relies on former experiences in order to choose the action with highest quality and the exploitation phase becomes dominant. However, in some self-optimization algorithms (like the one applied in this thesis) the exploration phase never ends and in a small portion of the time a random action is chosen instead over the action with highest quality in order to keep exploring new possible best actions for each sub-state.

Assume that after convergence of quality values, the best action for each sub-state and the corresponding value is as follows (shown values are made up numbers just to provide an overall insight about the methodology):

- $\text{Max}_l\, Q(s_{11}, a_l) = Q(s_{11}, a_7) = 0.75$
- $\text{Max}_l\, Q(s_{12}, a_l) = Q(s_{12}, a_7) = 0.61$
- $\text{Max}_l\, Q(s_{13}, a_l) = Q(s_{13}, a_5) = 0.80$
- $\text{Max}_l\, Q(s_{14}, a_l) = Q(s_{14}, a_6) = 0.87$

- $\mathrm{Max}_l\, Q(s_{21}, a_l) = Q(s_{21}, a_2) = 0.90$
- $\mathrm{Max}_l\, Q(s_{22}, a_l) = Q(s_{22}, a_7) = 0.69$
- $\mathrm{Max}_l\, Q(s_{23}, a_l) = Q(s_{23}, a_4) = 0.58$
- $\mathrm{Max}_l\, Q(s_{24}, a_l) = Q(s_{24}, a_4) = 0.70$
- $\mathrm{Max}_l\, Q(s_{31}, a_l) = Q(s_{31}, a_3) = 0.91$
- $\mathrm{Max}_l\, Q(s_{32}, a_l) = Q(s_{32}, a_2) = 0.83$
- $\mathrm{Max}_l\, Q(s_{33}, a_l) = Q(s_{33}, a_7) = 0.77$
- $\mathrm{Max}_l\, Q(s_{34}, a_l) = Q(s_{34}, a_4) = 0.64$
- $\mathrm{Max}_l\, Q(s_{41}, a_l) = Q(s_{41}, a_3) = 0.93$
- $\mathrm{Max}_l\, Q(s_{42}, a_l) = Q(s_{42}, a_2) = 0.88$
- $\mathrm{Max}_l\, Q(s_{43}, a_l) = Q(s_{43}, a_2) = 0.67$
- $\mathrm{Max}_l\, Q(s_{44}, a_l) = Q(s_{44}, a_7) = 0.80$

Referring to this look-up table, the controller agent has enough information to calculate the optimal action A(s(t)). The system state s(t) has been mapped to some sub-states with membership degrees and the best actions for those sub-states are known. Now the agent can make the decision and derive the action.
The taken action would be a combination of best actions in sub-states s_{12} , s_{13} , s_{22} and s_{23} or with probability of ε (experimenting rate) a random action to explore the potential of other actions; this means that with probability of ε, the controller chooses a random action out of all the possible actions instead of referring to the look-up table and select the action with the highest quality.
This experimentation phase is necessary because there might exist actions in a specific sub-state that have been chosen in former experiences while only the quality of those actions get updated and the other actions never have the chance of being chosen in that sub-state, although they might be very suitable. The derived action (A(s(t))) is calculated as the appropriately weighted average of the selected sub-state specific actions as below:

$$A(s(t)) = round\{\sum_{i=1}^{4}\sum_{j=1}^{4}\mu_{ij}(s(t))\cdot a^*(\mathrm{s_{ij}})\}$$

$$a^*(\mathrm{s_{ij}})\begin{cases} -\ \arg\max_a Q(s_{ij}, a) & \text{with probability of } 1-\varepsilon \\ -\ a_l \text{ where } l = \text{random } \{1,2,...,7\} & \text{with probability of } \varepsilon \end{cases}$$

In the continuation of this chapter we assume that $a^*(\mathrm{s_{ij}})$ is equal to arg $\max_a Q(s_{ij}, a)$ and the algorithm works in exploitation phase, i.e. that the best action rather than a random action is selected. In the given numerical example, the derived action (A(s(t)) is calculated as following:
$A(s(t)) = round\{ (\mu_{12} \times a_7) + (\mu_{13} \times a_5) + (\mu_{22} \times a_7) + (\mu_{23} \times a_4)\} =$

round{(0.19 × 0) + (0.05 × -2) + (0.16 × 0) + (0.15 × -1)} = round{ - 0.25} = 0
Note that - 0.25 is rounded off to the closest integer which is zero and means that the C_{Shared} value is not changed. After applying A(s(t)) to the system, the controller agent should wait for an observation time interval (Δt) to receive Q(s(t+Δt), A(s(t+Δt))) and r(s(t), A(s(t)) as the feedback of the taken action and update the quality of (sub-state , action) pairs according to the observed performance by the induced action.

$$\hat{Q}(s(t), A(s(t))) = \sum_{i=1}^{4}\sum_{j=1}^{4}\mu_{ij}(s(t))\cdot Q(s_{ij}, a^{*}(s_{ij}))$$

For the given example, $\hat{Q}$(s(t) , A(s(t))) is computed as:
Q(s(t) , A(s(t))) = μ_{12} × q (s_{12} , a_7) + μ_{13} × q (s_{13} , a_5) + μ_{22} × q (s_{22} , a_7) + μ_{23} × q (s_{23} , a_4) = 0.19 × 0.61 + 0.05 × 0.8 + 0.61 × 0.69 + 0.15 × 0.58 = 0.67,
which indicates the quality of the action A(s(t)) in the system state of s(t).
For calculating $\hat{Q}$(s(t) , A(s(t))) and r(s(t) , A(s(t)), the controller agent has to wait until the end of next observation time interval (t+Δt) because the new P_{Low} and P_{High} are required as feedback to assess the taken action performance. Assume at the end of the next observation time interval (t+Δt) the state of system is calculated and reported to the agent as

- P_{Low} (t+Δt) = 0.2
- P_{High} (t+Δt) = 0.3

$\longrightarrow$ s(t+Δt) = (0.2 , 0.3)

The controller agent can use this new state (s(t+Δt)) to calculate $\hat{Q}$(s(t),A(s(t))) and r(s(t),A(s(t)) and update the quality value entries of the (sub-state,action) look-up table. Two main aspects of evaluating the performance quality of the taken action are:

- *Short-term reward*
  How successful is the taken action A(s(t)) to maximize the reward function. This parameter is called the immediate reward r(s(t),A(s(t))).
- *Long-term reward*
  How high is the maximum possible quality of the system after the taken action given by ($\hat{Q}$(s(t),A(s(t)))). This parameter is important because it considers the impact of the action on future expecting rewards (r(s(t+Δt)), r(s(t +2Δt)),…). For example there might be an action which brings a high r(s(t),A(s(t))) but leads the system to future states (s(t+Δt), s(t+2Δt),…) with low expected rewards. So, even though an action brings some immediate rewards to the system, it may not be a very suitable action in the long run.

The immediate reward for the given example is computed below:
r(s(t),A(s(t))) = - P_{Low} (t+ Δt) - βP_{High} (t+ Δt) = - 0.2 - 10 × 0.3 = - 3.2
The future quality will be calculated according to the formula below:

$$\hat{Q}(s(t),A(s(t))) = \sum_{i=1}^{4}\sum_{j=1}^{4}\mu_{ij}(s(t+\Delta t))\cdot a^{*}(s_{ij})$$

For calculating $\hat{Q}$(s(t),A(s(t))), the membership degrees of new P_{Low} and P_{High} should be computed.

$$P_{Low}(t+\Delta t) = 0.2 \longrightarrow \left.\begin{array}{l} \frac{20-0}{33-0} \approx 0.6 \\ \frac{33-20}{33-0} \approx 0.4 \end{array}\right\} \begin{array}{l} \text{Membership degree of } P_{Low} \text{ to sub-state } s_2 = 0.6 \\ \text{Membership degree of } P_{Low} \text{ to sub-state } s_1 = 0.4 \end{array}$$

$$P_{High}(t+\Delta t) = 0.3 \longrightarrow \left.\begin{array}{l} \frac{30-0}{33-0} \approx 0.9 \\ \frac{33-30}{33-0} \approx 0.1 \end{array}\right\} \begin{array}{l} \text{Membership degree of } P_{Low} \\ \text{to sub-state } s_2 = 0.9 \\ \text{Membership degree of } P_{Low} \\ \text{to sub-state } s_1 = 0.1 \end{array}$$

Membership degree vectors of the new P_{Low} and P_{High} are as below:
$\mu_{High} = (0.1, 0.9, 0, 0)$ and $\mu_{Low} = (0.4, 0.6, 0, 0)$.
Having new membership degree vectors, new membership degree matrix is computed as:
$\mu_{11}(s(t+\Delta t)) = 0.04$
$\mu_{12}(s(t+\Delta t)) = 0.06$
$\mu_{21}(s(t+\Delta t)) = 0.36$
$\mu_{22}(s(t+\Delta t)) = 0.54$
while the rest of $\mu_{ij}(s(t+\Delta t)) = 0$.
Referring to the look-up table, the highest expected quality of the new (sub-state, action) pairs can be found. Having all the information, the next step would be to calculate the expected future quality.

$\hat{Q}(s(t+\Delta t),A(s(t+\Delta t))) = \boldsymbol{\mu}_{11} \times q(s_{11}, a_7) + \boldsymbol{\mu}_{12} \times q(s_{12}, a_7) + \boldsymbol{\mu}_{21} \times q(s_{21}, a_2) + \boldsymbol{\mu}_{22} \times q(s_{22}, a_7) = 0.04 \times 0.75 + 0.06 \times 0.61 + 0.36 \times 0.9 + 0.54 \times 0.69 = 0.76$

Knowing immediate reward $(r(s(t),A(s(t)))$ and future quality $\hat{Q}(s(t),A(s(t)))$, all the required information for updating the (sub-state , action) quality look-up table is available.
For all the sub-states (s_{ij}) :
$Q_{new}(s_{ij}(t), a^*(s_{ij}(t))) = Q_{old}(s_{ij}(t), a^*(s_{ij}(t))) + \alpha\, \Delta Q \,.\, \mu_{ij}\left((s_{ij}(t)\right)$
Where
$\Delta\hat{Q} = r(s(t+\Delta t)) + \gamma\, \hat{Q}(s(t+\Delta t),A(s(t+\Delta t))) - \hat{Q}(s(t),A(s(t)))$
In the designed fuzzy-Q learning algorithm, the discount factor (γ) is set to 0.8. So, $\Delta\hat{Q}$ is calculated as:
$\Delta\hat{Q} = -3.2 + 0.8 \times 0.76 - 0.67 = -3.26$.
The learning rate α indicates to what extent this ΔQ is going to be applied to update the present quality look-up table entries. The applied value of α is 0.8.
The last step is applying $\alpha \times \Delta Q$ to the look-up table and updating the quality values considering the membership degrees.
In summary the steps of the designed fuzzy-Q Learning algorithm have been described as follows:
1. Initializing the Q-look-up table $(s_{ij},a_l = 0)$ and time $(t = 0)$.

2. Receiving the system state $\mathbf{s}(t) = (P_{Low}(t), P_{High}(t))$ every observation time interval (Δt)

3. Discretizing and mapping the received state to the sub-states

4. Computing membership vectors and matrix.

5. Referring to the (sub-state,action) quality look-up table and deriving the best actions for each sub-state.

6. Calculating the inferred action.

7. Calculating corresponding quality of the taken action.

8. Executing the action A(s(t))

9. t = t+Δt and A(s(t)) leads the system to the state s(t+Δt).

10. Receiving the immediate reward (r(s(t),A(s(t)))).

11. Calculating the highest possible future quality ($\hat{Q}$ (s(t+Δt) , A(s(t+Δt))).

12. Calculating $\Delta\hat{Q}$.

13. Updating the Q look-up table by the factor $\alpha \times \Delta\hat{Q}$.

14. Go back to step 2.

Figure 3-4 illustrates the time scheduling in the fuzzy-Q learning algorithm.

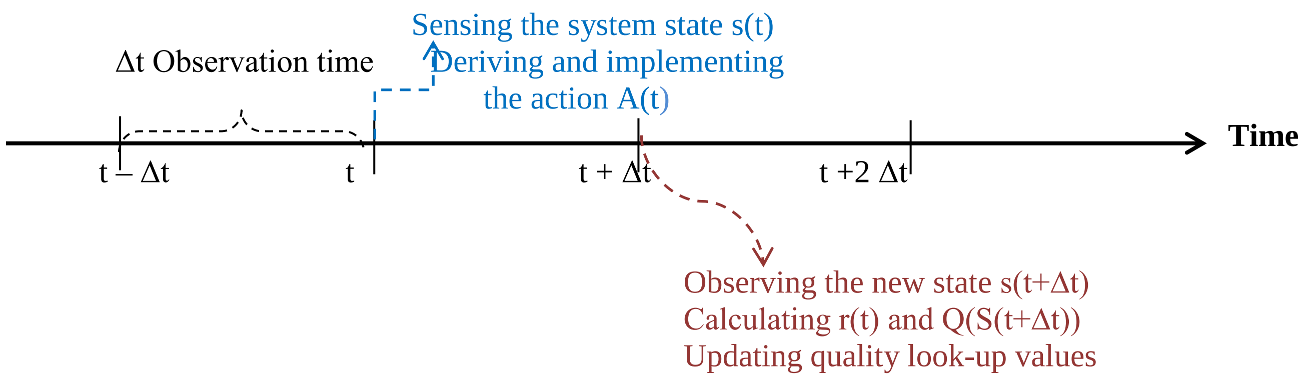

Sensing the system state s(t)
Deriving and implementing
the action A(t)
Δt Observation time
Time
t – Δt
t
t + Δt
t +2 Δt
Observing the new state s(t+Δt)
Calculating r(t) and Q(S(t+Δt))
Updating quality look-up values

Figure 3.4: Describing the time scheduling of the Fuzzy-Q Learning method

3.3.3 Design Choices and Considerations

In designing and applying the self-optimization algorithms, a lot of design choices need to be made and parameters need to be set involving all the tradeoffs and performance concerns. This section provides a concise overview of these design considerations and their qualitative effects on the overall performance of the algorithm. Two first design choices are common for both of the algorithms while the rest of the design considerations is specified for the fuzzy-Q learning method.

3.3.3.1 Design Choices and Considerations for Both of Algorithms

- The first essential choice in self-optimization algorithms is defining the reward function (r(s(t),A(s(t))) according to the self-optimization objectives and the inherent characteristics of the methodologies. The objective of the system should be carefully translated to the reward, e.g. the identical objective of minimizing the long-term Blocking in our case study is translated to balancing P_{Low} and P_{High} in the way that

$P_{Low}/ P_{High} = \beta$ as the objective of rule-based method while the objective of fuzzy-Q learning algorithm is maximizing a discounted cumulative reward.

- Observation time interval (Δt) should be chosen considering λ (expected occurrences per time). Δt should be large enough to observe enough occurrences in order to provide reliable statistics as P_{Low} and P_{High}. On the other hand, a very long Δt leads in to a slow controller agent. The controller agent must wait for Δt for taking a new action and react to the system state changes.

3.3.3.2 Design Choices and Considerations for Fuzzy-Q learning

- One of the first choices in designing algorithms is the number of sub-states (fuzzification). Defining more sub-state provides the controller agent the facility of mapping the reported state to sub-states more accurately. For example, assume instead of four sub-states for blocking probabilities, ten sub-states were designed. With four defined sub-states, $P_{Low} = 0.1$ and $P_{Low} = 0.3$ are both mapped to s_1 and s_2 while in the case of defining ten sub-states they are mapped to totally different sub-states. This means that the controller agent has more detailed information about the system state and can act more accurately based on this information. The drawback of having more sub-states is having more Q(sub-state,action) entries and having a larger quality look-up table increases the time needed for convergence of the quality values. In other words, the larger quality look-up table needs a longer exploration phase before converging to reliable values which can be used in exploitation phase.
- The number of state indicators is another design choice. By adding more quality indicators, the system state is described better for the controller agent and this additional information can assist the controller agent deciding more accurately, e.g. assume that the number of arrivals for each class is added to the blocking probabilities. In that case the blocking probability values are more meaningful because the controller can consider the arrival process characteristics and makes a better judgment about the optimal C_{Shared}. Again the disadvantage of having more state indicators is increasing the size of quality look-up table and consequently increasing the convergence time of the quality entries.
- The number and kind of actions defined for each state is another matter of design in fuzzy-Q learning algorithm. More actions provides the controller agent the better facilities to enhance its performance. Defining more actions for each state, increase the number of Q(sub-state,action) and the larger quality look-up table, makes the system slow towards transition from exploration phase to the exploitation phase. Another design consideration related to actions is the choice of designing the proper set of actions, e.g. where the actions are relative (like increasing C_{Shared} by one channel) or determined (e.g. set $C_{Shared} = 30$).
- Parameter setting choices in fuzzy-Q learning algorithm is setting the discount factor (γ). The discount factor indicates the relevant importance of the future reward over the instantaneous reward. Setting the γ close to zero indicates that instantaneous reward is much more important than future reward while a γ close to one means that the future

rewards are as valuable as achieving high immediate reward. Setting $\gamma = 0.8$ indicates that for our system receiving high rewards in the future is almost as important as immediate reward.

- The learning rate (α) is a factor indicating to what extent the received feedback of a taken action changes the quality of that (sub-state, action) value. If α is close to 1, the calculated judgments is directly applied in adapting the quality values while α close to zero means that the procedure of updating the quality values is slow. α is set to 0.8 in our algorithm.
- The experimenting rate (ε) is another parameter that needs to be set. ε indicates how often a random action is chosen (exploration phase) instead of choosing the action with highest quality value (exploitation phase). A non-zero ε is necessary to keep the exploration phase running. The case of $\varepsilon = 0$ does not allow the controller agent to explore new actions and forces it to always pick the action with highest quality. Note that at the beginning ($t = 0$), all the (sub-state,action) qualities are zero and there is no action with highest quality in which case random actions are chosen. $\varepsilon = 1$ means that the controller agent always chooses a random action and never uses the learnt information based on former experiences which effectively leads to an inefficient self-optimization method which is just taking random actions.

Table 3-3 gives an overview of the key design choices and the associated tradeoffs for the designed fuzzy-Q learning algorithm.

Name of the parameter	***Chosen value***	***Designing consideration***
Number of sub-states for each blocking probability	4	More sub-states +: Higher accuracy −: larger quality look up table, slower quality values convergence
Number of possible actions for each sub-state	7	More actions +: more chances for taking precise action −: larger quality look up table, slower quality value convergence
Observation time interval: Δt	100 s	Larger Δt +: more reliable statistics of blocking probabilities −: slower control agent reaction to the system state

Discount factor: γ (0 < γ <1)	0.8	Higher γ +: investigates on long-term rewards in the future −: assigning less value to the immediate quality of the taken action
Learning rate: α (0 < α <1)	0.8	Higher α +: direct applying of the feedbacks, faster reaction to the system state −: fluctuating quality values
Experimenting rate: ε (0 < ε <1)	0.2	Higher ε +: more often taking random actions to explore, exploring new potential beneficial actions −: exploration is inherently costly and the algorithm may shows bad performance during the exploration because of taking a wrong action by random

Table 3-3: An overview of design algorithm choices

4 Simulation Scenarios and Results

In this chapter, I will define three sets of simulations that are applied to a simulator developed in Delphi 5 (the main unit of the code is provided in Appendix B). The objective of the first group of simulations is to study the effect of different parameters' values on the overall performance of self-optimization algorithms. The second set of simulation scenarios will replicate a partial failure in the total capacity of the system (C_{Total}) as well as repairing the failure allowing the author to observe the algorithm's reaction to this abrupt change in capacity. The last group of simulations is modeling changes in arrival process profile.
In all of the aforementioned scenarios, the self-optimization algorithm's performance is assessed by the following quality matrix:

- ***Convergence time of cumulative Blocking***
- ***Converged cumulative Blocking value***
- ***90% of P_{Low} after convergence***
- ***90% of P_{High} after convergence***
- ***Time fraction of meeting P^*_{Low} and P^*_{High} after convergence***

Convergence time of cumulative Blocking: This entry of the quality matrix represents the time of convergence of the cumulative Blocking ($P_{Low} + 10P_{High}$) values. Cumulative Blocking at time *t'* is the Blocking from *t=0* to *t=t'*. $Blocking(t')=P_{Low}(t=0 \text{ to } t') + 10P_{High}(t=0 \text{ to } t')$.
The cumulative reward has converged if the difference between seventy consecutive Blocking values is less than 0.0001, which means that cumulative Blocking values are almost identical and the Blocking has reached its converged value.
Converged cumulative Blocking value: Assuming the same definition of the convergence, this term represents the converged value of the cumulative Blocking.
***90% of P_{Low} and P_{High} after convergence of cumulative Blocking*:** in 300 seconds (five minutes) interval P_{Low} and P_{High} are measured for more reliable statistics. These statistical blocking probability values are sorted in ascending order and the 90% value of all values is extracted. These values indicate that in 90% of the five minutes statistical intervals, P_{Low} and P_{High} are lower than given values. These values provide operational statistics for the operator. The operator can claim to guarantee lower blocking probabilities in 90% of the time.
***Time fraction of meeting P^*_{Low} and P^*_{High} after convergence*:** P^*_{Low} and P^*_{High} are targeted values defined by the operator. The system performance is considered acceptable if $P_{Low} < P^*_{Low}$ and $P_{High} < P^*_{High}$. This entry of the quality matrix shows the time fraction in which the system is successful

meeting the $P_{Low} < P^*_{Low}$ and $P_{High} < P^*_{High}$ requirement. For all the simulations $P_{Low} = 10\%$ *and* $P_{High} = 1\%$.

All the simulations duration is equal to 30000 × Δt(Observation time) .

4.1 Changing in Parameters Scenarios

The goal of this series of simulations is studying the effect of algorithms' parameter settings on the overall system performance.

4.1.1 Default Setting

A default setting for both algorithms is defined as following:
$C_{Total} = 35$
C_{Shared} initialization = 10
λ (expected occurrences per time unit) = 30
Average job duration = 0.8
Percentage of high priority jobs = 60%
β (Relative importance of high priority job over low priority job) = 10
Observation time interval = 100 seconds
Statistical time interval = 300
$P^*_{Low} = 10\%$
$P^*_{High} = 1\%$
Additional parameter settings are needed for Fuzzy-Q Learning algorithms such as:
γ (Discount Factor) = 0.8
α (Learning Rate) = 0.8
ε (Experimenting Rate) = 0.2
The quality matrix values that resulted from the default setting simulations for rule-based and Fuzzy-Q Learning method are recorded in Table 4-1 and the performance of the algorithms are shown in Figure 4-1 and 4-2.

Self-optimization method	**Converged cumulative Blocking value**	**Convergence time (Seconds)**	**Time fraction of meeting P_{Low} and P_{High} target**	**90% of P_{High} after convergence**	**90% of P_{Low} after convergence**
Rule-based method	0.0669866138	83400	0.999691	0.004603	0.036192
Fuzz-Q learning method	0.1024201422	147400	0.88273	0.007543	0.097688

Table 4-1: The quality matrix of default setting scenario

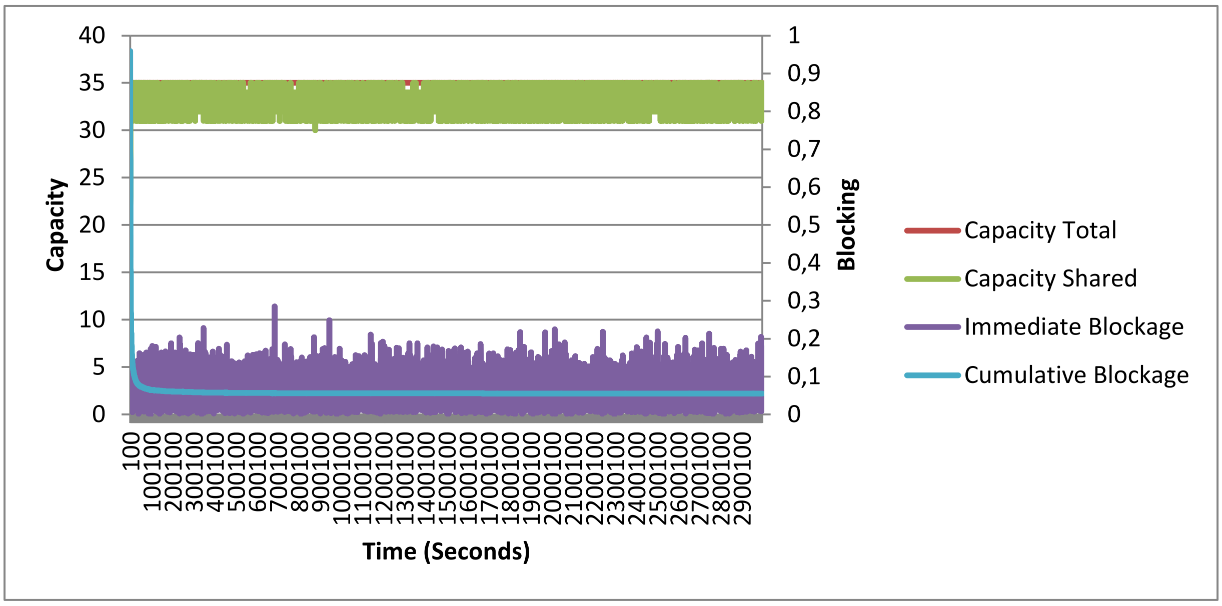


Figure 4.1: Rule-based method performance in the default setting

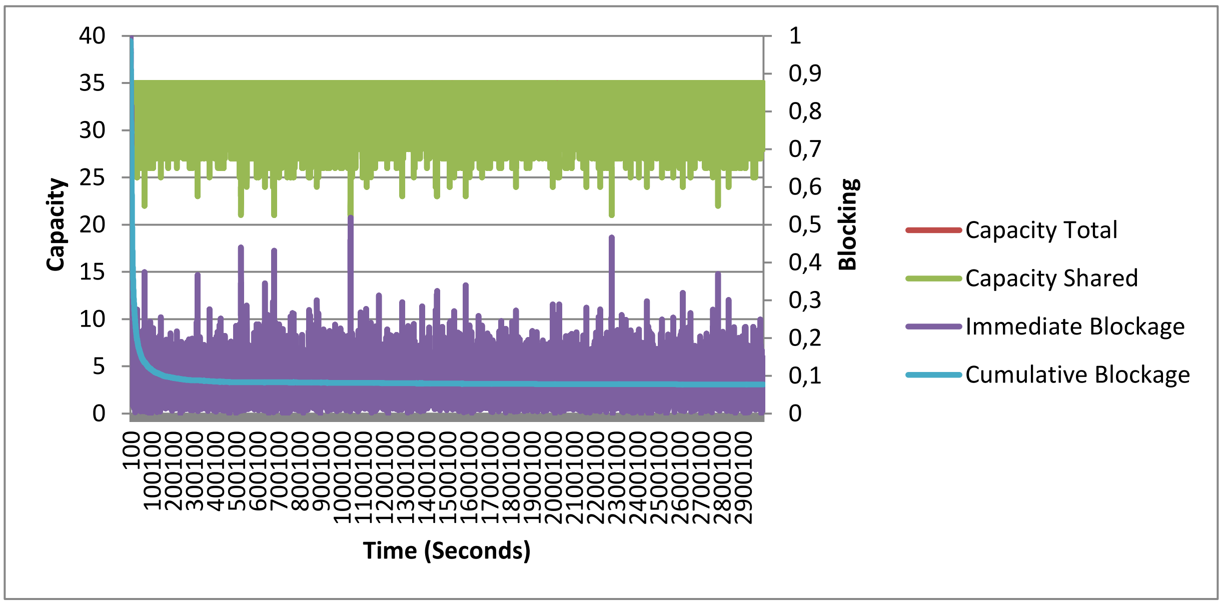


Figure 4.2: Fuzzy-Q learning method performance in the default setting

According to the above figures, the rule-based method has performed better (faster convergence, lower converged cumulative blocking and larger time fraction of meeting targeted Blockings) with higher stability because it does not have a learning process or an experimenting phase. In order to compare both methodologies, cumulative Blocking of methods is plotted on Figure 4-3.

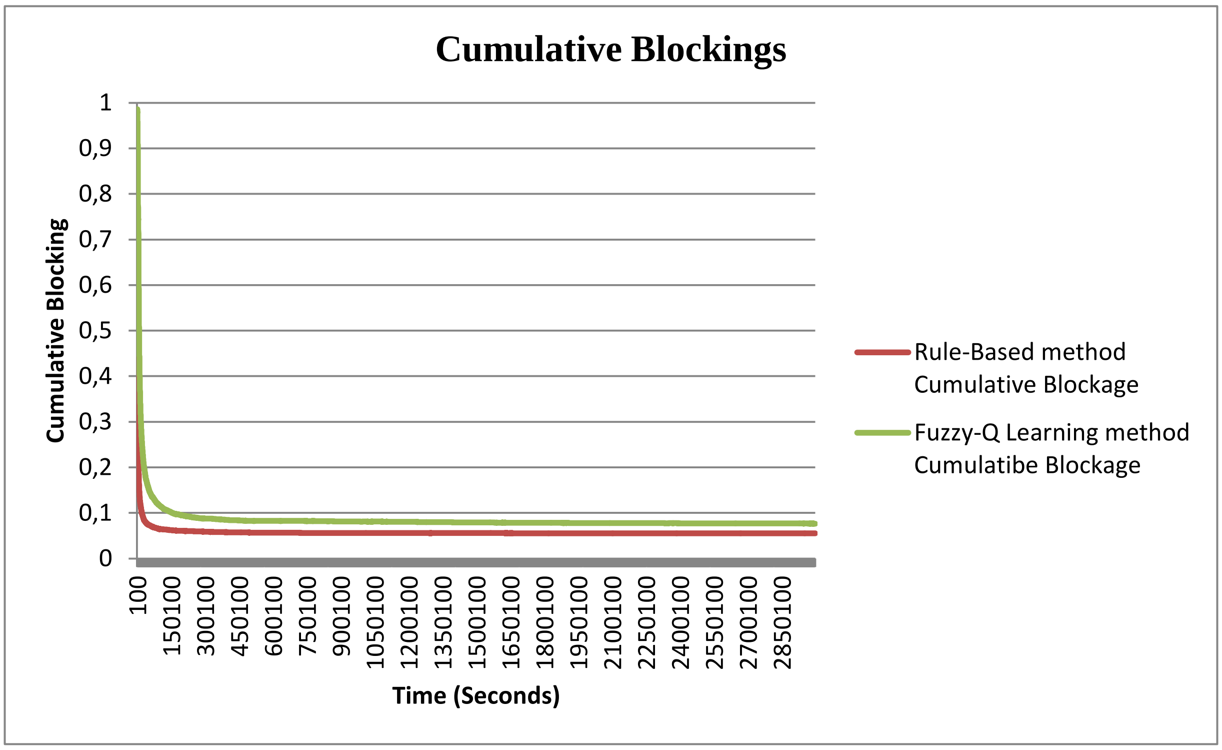


Figure 4.3: Comparing Cumulative Blockings of both methods

In the beginning of the simulation, the rule-based has performed better because it knows the optimal action while the Fuzzy-Q learning must learn about the best action to take. However the overall cumulative Blocking of the fuzzy-Q learning method is slightly higher than that one of the rule-based method but both of the methods successfully converge quickly to the optimal C_{Shared}.

4.1.2 Change in Observation Time (Common for Both of Algorithms)

The goal of the next set of the scenarios is to monitor the observation time length (Δt) on the overall performance of algorithms. The observation time is set to five different values and the resulting quality matrix for each self-optimization algorithm is shown in Table 4-2 (rule-based) and table 4-3(fuzzy-Q learning):

Observation time	Converged cumulative Blocking value	Convergence time (Seconds)	Time fraction of meeting P_{Low} and P_{High} target	90% of P_{High} after convergence	90% of P_{Low} after convergence
2 seconds	0.1027074456	3820	0.84492	0.0014603	0.1035791

50 seconds	0.0669935516	45300	0.999794	0.004809	0.035396
Default Setting (100 seconds)	0.0669866138	83400	0.999691	0.004603	0.036192
500 seconds	0.0703425341	288500	0.999715	0.004671	0.038535
1000 seconds	0.0724399872	500000	0.999786	0.005339	0.043087

Table 4-2: The resulting quality matrix for different observation interval sizes in the rule-based method

Observation time	**Converged cumulative Blocking value**	**Convergence time (Seconds)**	**Time fraction of meeting P_{Low} and P_{High} target**	**90% of P_{High} after convergence**	**90% of P_{Low} after convergence**
2 seconds	0.1336444671	5630	0.59638	0.004855	0.18608
50 seconds	0.1087111680	73100	0.877782	0.007284	0.102345
Default Setting (100 seconds)	0.1024201422	147400	0.88273	0.007543	0.097688
500 seconds	0.1118317622	641000	0.90103	0.007661	0.087248
1000 seconds	0.1107673098	951000	0.876921	0.009417	0.128128

Table 4-3: The resulting quality matrix for different observation interval sizes in the fuzzy-Q learning method

The above tables illustrate the fact that the length of observation time has a direct effect on convergence time. The shorter Δt leads to a shorter convergence time, a fact that shows that the system finds the optimal CShared more quickly. However, if the observation time is as short as two seconds, the calculated blocking probabilities are not reliable because they are based on too few occurrences. That's the reason that both of the algorithms with $\Delta t = 2$ seconds shows the worst performance of all the scenarios. A long observation time, on the other hand, leads to a slower system reaction to the changes in the environment. Long observation time provides more reliable statistical information for the controller agent, but in the case of a change in the environment or of taking inappropriate action, the controller reacts slowly and the algorithm performance remains bad for a longer time. The higher converged cumulative Blocking value for longer observation time is the consequence of the longer time that the controller agent spends to find the optimal C_{Shared} as well as not reacting fast enough to the changes of the system state. I have plotted the three most important quality matrix entries in Figures 4-4 to 4-6 (90% P_{Low} and 90% P_{High} after convergence figures in the Appendix A) to provide a clear comparison between algorithms.

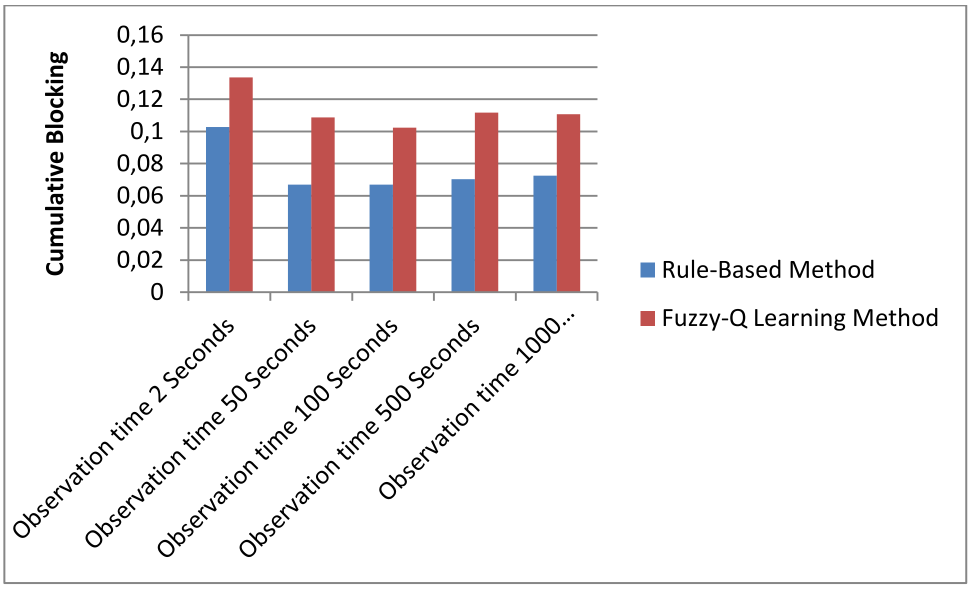


Figure 4.4: Comparing converged cumulative Blocking values in both methods

As illustrated in Figure 4-4, the cumulative Blocking of the fuzzy-Q learning method in all the scenarios is higher than that of the rule-based method, while the relative difference between them is almost the same. It is also remarkable that for both algorithms, observation time set to 100 seconds has the lowest cumulative reward of all the scenarios. We can conclude that regardless of the applied self-optimization method, an appropriate observation time should be chosen according to the arrival process characteristics (λ).

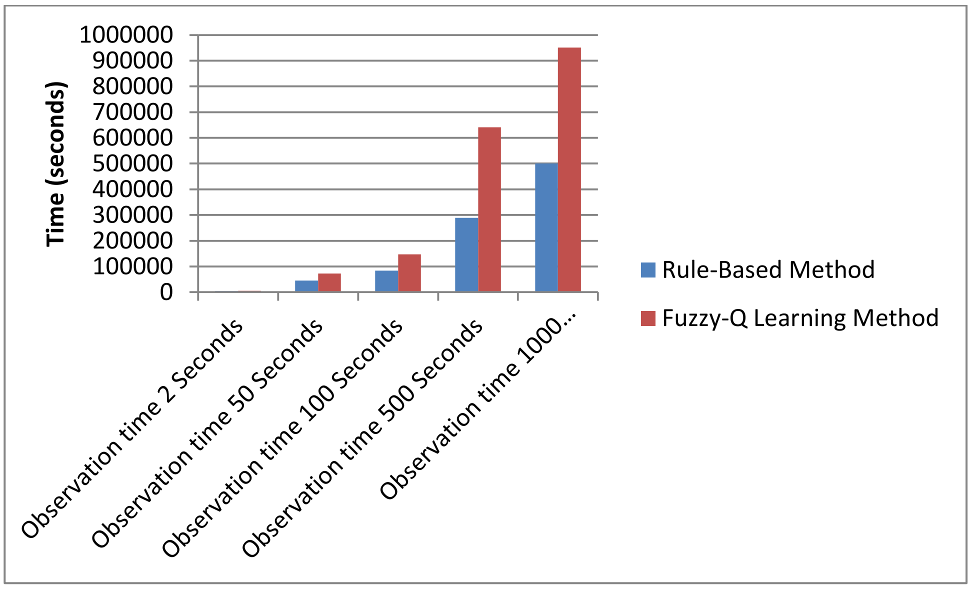


Figure 4.5: Comparing cumulative Blocking convergence time in both methods

Figure 4-5 evidences that in all the scenarios, the fuzzy-Q learning method has converged later than the rule-based method, but the differences between two methods' convergence time grows when the observation time is increased. This is not only because the system is acting slowly to converge to the optimal C_{Shared}; it is also the result of the fact that the learning phase takes longer. In the rule-based method, the controller agent already knows the proper action but it must wait to take the action after finishing the observation time (1000 seconds), while in the fuzzy-Q learning method the controller agent finds out 1000 seconds later if the action is an inappropriate one and should start exploring a new action to perform better in that state. Thus, both the action taking process and exploration phase are longer, which consequently delay the convergence time in the fuzzy-Q learning more than the rule-based method.

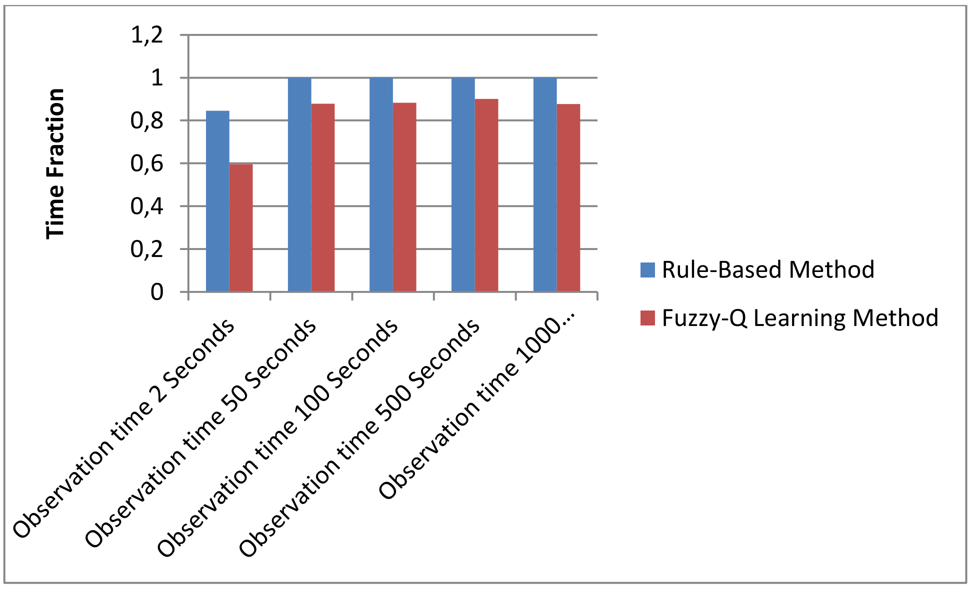


Figure 4.6: Comparing time fraction of meeting targeted Blocking (P_{Low} < 10% and P_{High} <1%) after convergence

As it is evidenced in Figure 4-6, the observation time set to 2 seconds shows worse performance than other scenarios. After convergence the time fraction of meeting targeted Blocking is almost identical and, in all the cases, the fuzzy-Q learning method time fraction is lower than the rule-based method because of the learning process and experimenting phase. Despite that the fuzzy-Q learning still performs acceptably with meeting targeted Blocking in more than 80% of the time.

The time fraction of meeting P^*_{Low} and P^*_{High} in the first twenty hours for different observation interval size is plotted in Figures 4-7 and 4-8 in order to provide a better visualization of the effect of observation time length on the system performance before the convergence.

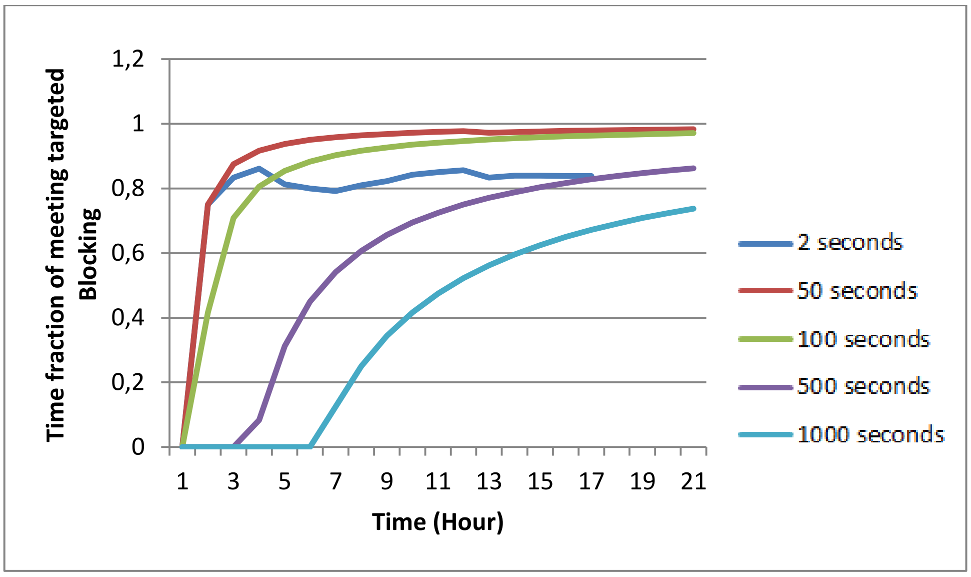


Figure 4.7: Rule-based algorithm performance for different Δt in the 20 hours of running the simulation

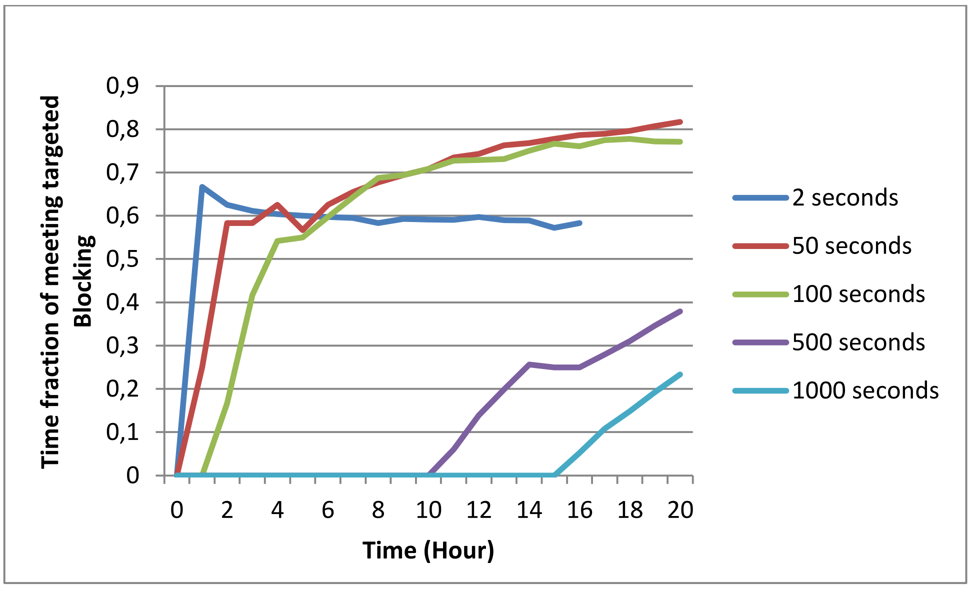


Figure 4.8: Fuzzy-Q learning algorithm performance for different Δt in the first 20 hours of running the simulation

Both figures show that the longer observation time results in a delay in convergence. For example, Figure 4-7 illustrates that the rule-based algorithm with observation time set to 2, 20 and 100 seconds is converged to the optimal C_{Shared} value in less than three hours while in the scenario with $\Delta t = 1000$ it has not converged totally within twenty hours. It is also apparent that the observation time set to 2 seconds never performs as well as other settings even after fifteen hours of simulation.

Comparing Figures 4-7 and 4-8, it is evident that the overall time fraction of meeting targeted Blocking in fuzzy-Q learning algorithm is lower than that of the rule-based algorithm. The reason is that at the beginning of the simulation all of the possible actions have Q(sub-state, action) equal to zero and the controller agent should learn the best action in any sub-state by taking a particular action and adjusting based on the feedback. Even after convergence, the controller agent take a random action to explore new possible action (Experimenting Rate = 0.2) 20% of the time. This random action might be a very inappropriate one for the current state and lead to a lowering of the algorithm's performance.

4.1.3 Change in Parameter Scenarios for the Fuzzy-Q Learning Method

The rest of parameter changing scenarios are only applicable to the fuzzy-Q learning method. In the next set of simulation scenarios, I study the effect of different values of the discount factor, learning rate and experimenting rate on the general performance of the algorithm. In all of the scenarios, only one parameter value is changed and the rest of the parameters are set according to default settings.

The resulted quality matrix of this set of scenarios is presented in Table 4-4:

Simulation scenarios	**Converged cumulative Blocking value**	**Convergence time (Seconds)**	**Time fraction of meeting P_{Low} and P_{High} target**	**90% of P_{High} after convergence**	**90% of P_{Low} after convergence**
Discount Factor 0.2	0.1094196230	141800	0.810222	0.006473	0.121524
Discount Factor 0.5	0.1021094418	128800	0.842198	0.007031	0.111966
Discount Factor 0.8 (Default Setting)	0.1024201422	147400	0.88273	0.007543	0.097688
Discount Factor 0.9	0.1030199232	139100	0.892885	0.008028	0.088808

Learning Rate 0.2	0.1190116431	176700	0.927285	0.007643	0.078734
Learning Rate 0.5	0.1062706898	138900	0.901998	0.007762	0.088618
Learning Rate 0.8 (Default Setting)	0.1024201422	147400	0.88273	0.007543	0.097688
Learning Rate 0.9	0.1027510722	140300	0.871738	0.00755	0.101565
Experimentin Rate 0.2 (Default Setting)	0.1024201422	147400	0.88273	0.007543	0.097688
Experimentin Rate 0.5	0.1357384758	185600	0.801446	0.006897	0.131357
Experimentin Rate 0.9	0.9896108035	10500	0.0000	0.0000	1

Table 4-4: The resulting quality matrix for different parameter setting in the fuzzy-Q learning method

To provide more specific information about the effect of each parameter the convergence time, converged value of the cumulative Blocking and time fraction of meeting targeted Blocking are plotted in Figures 4-9 to 4-11 for all the scenarios including the default setting (figures of 90% of P_{Low} and P_{High} after convergence are available in appendix).

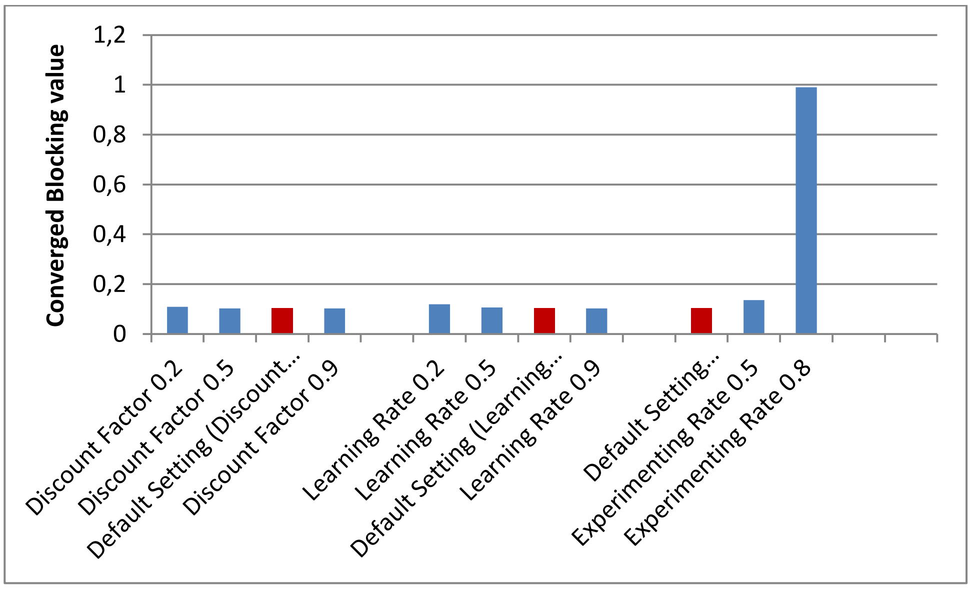


Figure 4.9: Converged value of the cumulative Blocking for all the simulation scenarios in the fuzzy-Q learning method

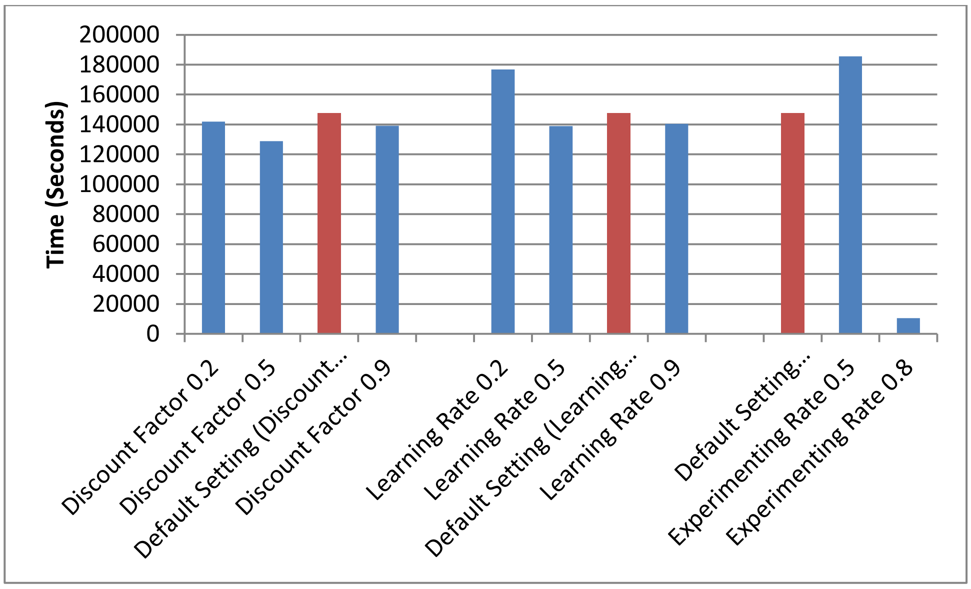


Figure 4.10: Convergence time of the cumulative Blocking for all the simulation scenarios in the fuzzy-Q learning method

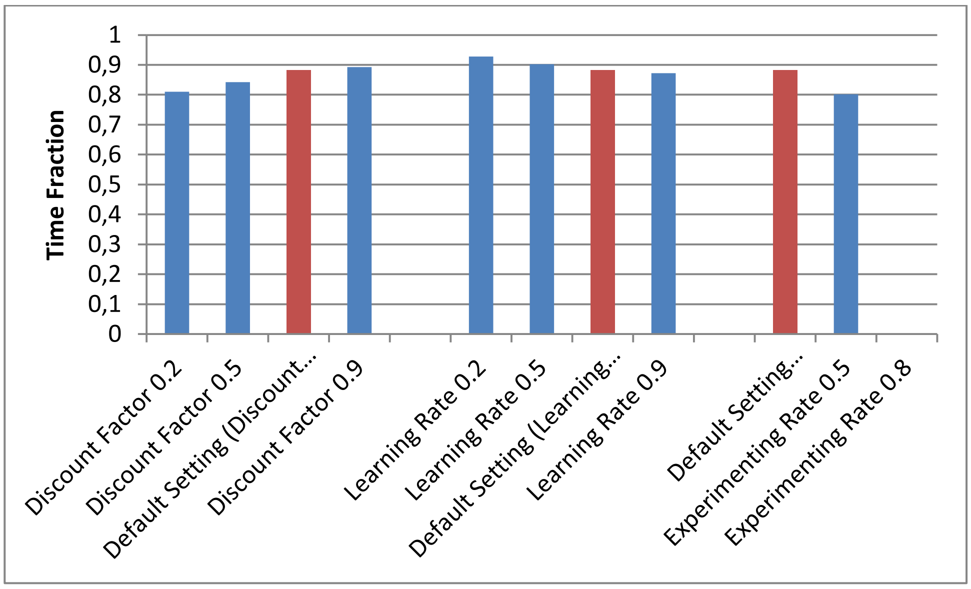


Figure 4.11: Time fraction of meeting targeted Blocking for all the simulation scenarios in Fuzzy-Q learning method

A higher discount factor considers the future reward as important as the immediate reward. The above figures show that the discount factor value does not have a considerable effect on convergence time and converged value, yet a higher discount factor shows better performance regarding meeting the targeted Blocking.
The lower learning rate indicates that the received feedback as a reward is implemented smoothly to change the Q(sub-state, action) values. Therefore, the (sub-state, action) quality matrix changes slowly towards the converged value. Results for different settings of the learning rate illustrate that the lower learning rate converges later but, performs slightly better in the case of meeting targeted Blocking.

The most dramatic changes in the algorithm performance are related to different settings of the experimenting rate. The experimenting rate indicates how often the systems chooses a random action rather than the action with highest quality (exploration phase). The experimenting rate = 0.9 means that in 90% of the time, the controller agent takes a random action. This high experimenting rate deteriorates the self-optimization algorithm's performance in the way that the system never meets targeted Blocking and the converged cumulative Blocking is almost 9 times higher than other scenarios. It perhaps worth mentioning that although the quality of performance with the experimenting rate set to 0.5 is lower than that of 0.2, the algorithm is still working reasonably.

4.2 Change in Capacity Scenarios

The goal of the second group of simulation scenarios is to analyze the reaction of self-optimization algorithms towards failure in capacity. In coordinated scenarios, at t= 1000000 seconds, a part of C_{Total} fails and the whole capacity is decreased to a new value. The system remains in this failure situation for 1000000 seconds and at t = 2000000seconds the capacity is repaired and the value of C_{Total} returns to the former value. In other words, the whole simulation time (3000000 seconds) is divided into three equal intervals. In the first and third interval, the system is working with C_{Total} = 35, while in the second interval the capacity total has decreased because of a partial failure. The self-optimization algorithm should sense this change, adapt to it and converges to a new value of C_{Shared} according to new characteristics of the system (new C_{Total}). Investigating algorithms' adaption to a failure in capacity is important because, in many cases, a failure in cloud's servers or cell's antennas may occur resulting in lower total capacity of the system, and a successful self-optimization algorithm should be able to adjust to new circumstances. In this set of simulation, self –optimization algorithms encounter situations of failing C_{Total} by 5, 10 and 20 channels.

The first assumption is failing Capacity by 5 channels. In Figures 4-12 (the rule-based method) and 4-13 (the fuzzy-Q learning method) the algorithms' responses to this failure are plotted.

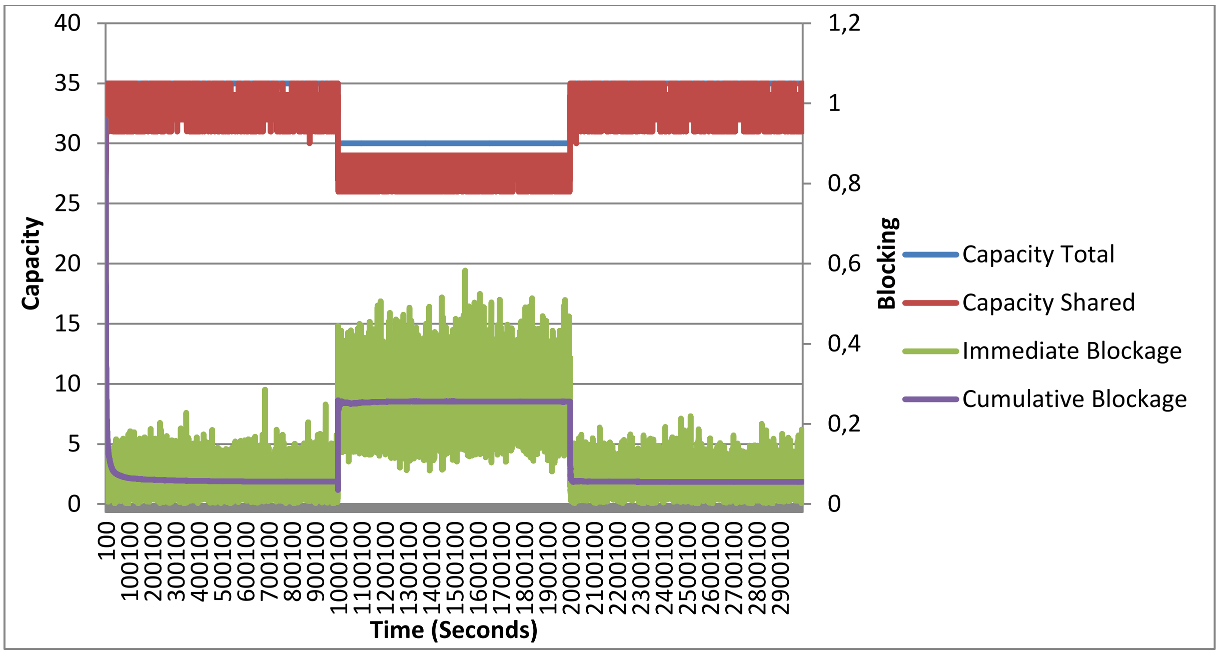


Figure 4.12: Rule-based method reaction to 5 channels failure of the C_{Total}

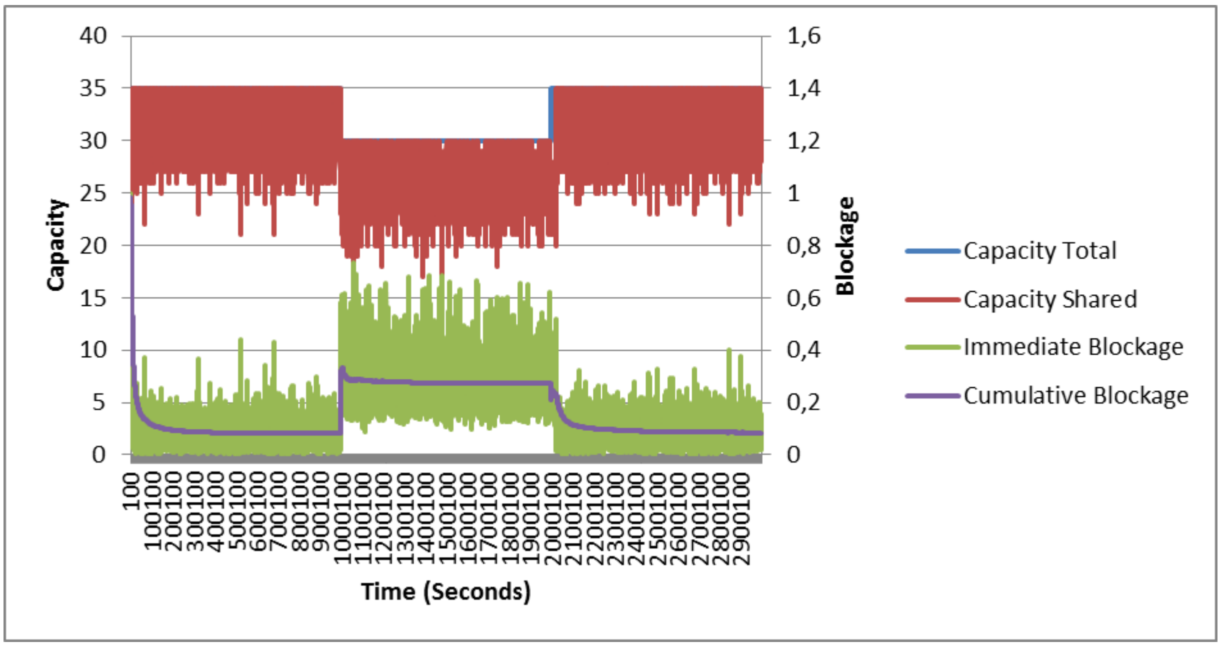


Figure 4.13: Fuzzy-Q learning method reaction to 5 channels failure of the C_{Total}

The above figures highlight the fact that both of the methods react to the change immediately and converge to the new C_{Shared} value at an appropriate speed. Fuzzy-Q Learning's performance shows more fluctuation in the converged C_{Shared} but the converged cumulative Blockings values are relatively similar.
The next failure scenario is a 10 channels failure in C_{Total}. The algorithms' performances are shown in Figure 4-14 and 4-15.

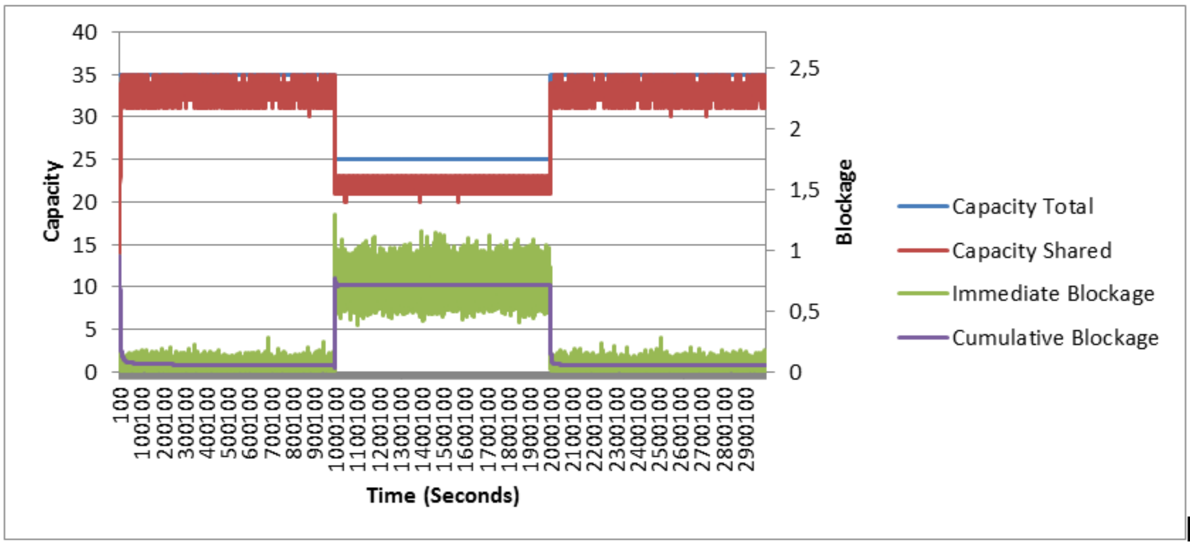


Figure 4.14: Rule-based method reaction to 10 channels failure of the C_{Total}

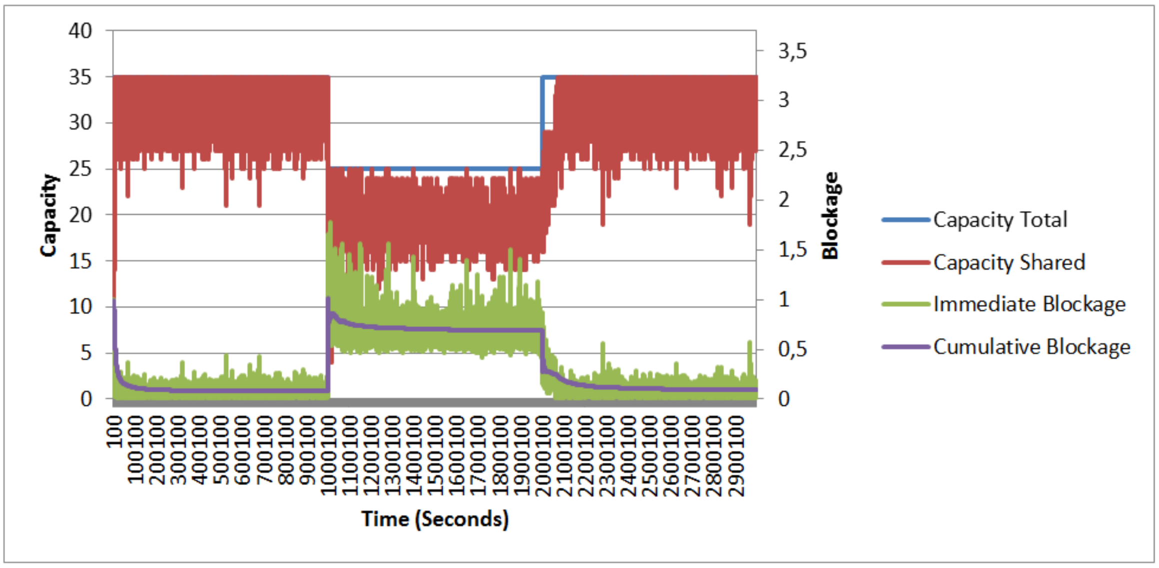


Figure 4.15: Fuzzy-Q learning method reaction to 10 channels failure in C_{Total}

In the case of a 10 channels failure in capacity, the fuzzy-Q learning method shows worse performance before convergence compared to rule-based method (representing by cumulative Blocking's slope). The reason is that a dramatic increase in blocking probabilities resulting from the failure in capacity forces the system to experience totally new states and, consequently the controller agent must learn which action is the best to take in these new states.

Another noticeable point is that the fuzzy-Q learning algorithm takes a longer period of time to find the optimal C_{Shared} value after repairing the failure compared to the rule-based algorithm. The decision making logic of the simulator can explain this phenomenon easily. If the derived action results in a C_{Shared} larger than total capacity of the system (which is the tendency of the system in the case of failure), C_{Shared} is set to C_{Total} value. Therefore, there is always a high limit for C_{Shared} and that makes the convergence process easier in the case of failure; however, in the case of repairing, the controller agent must converge to the new optimal C_{Shared} starting from the old optimal C_{Shared} without the help of forced limitation. This phenomenon also exists in the rule-based method's performance but it is not visible because in any case the convergence to the optimal C_{Shared} process for the rule-based method is too fast to be observed.

The last scenario of change in capacity is the case of occurring a failure in C_{Total} by 20 channels. Algorithms respond to this change presented in Figures 4-16 (the rule-based method) and 4-17(the fussy Q-learning method).

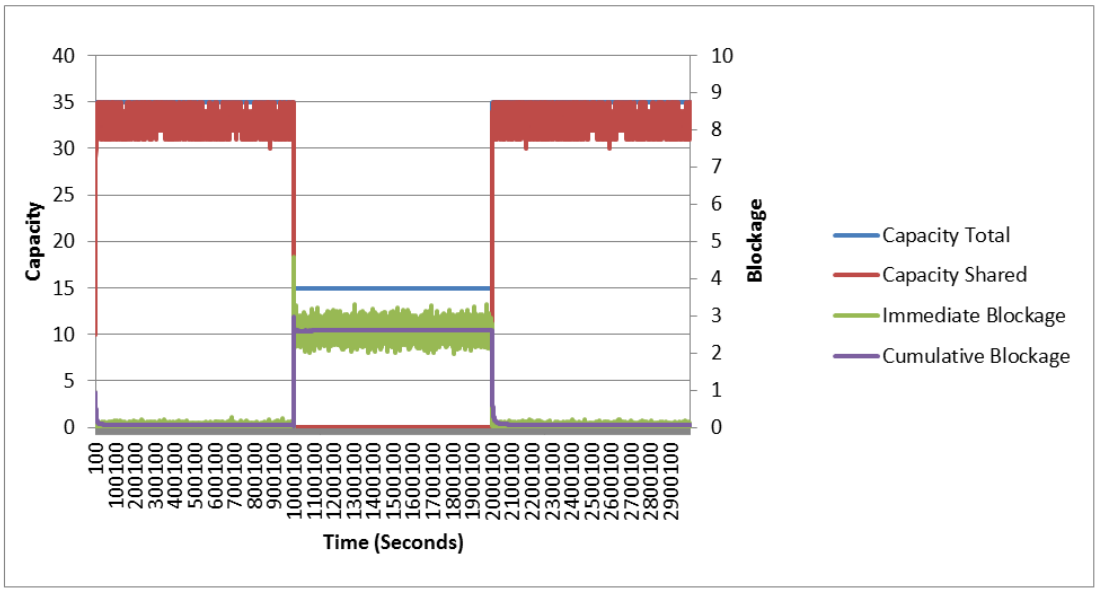


Figure 4.16: Rule-based method reaction to 20 channels failure of the C_{Total}

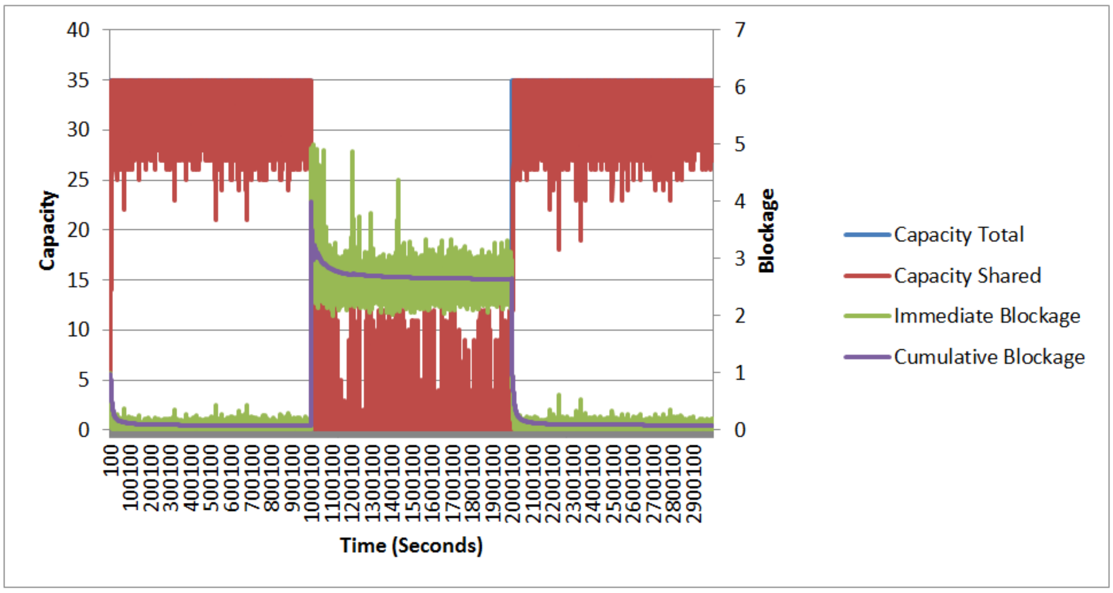


Figure 4.17: Fuzzy-Q learning method reaction to 20 channels failure of the C_{Total}

In the case of a 20 channels failure in capacity, blocking probabilities are too high for both low and high priority jobs. The rule-based method sets the C_{Shared} to zero which means the system is blocked for low priority jobs and only high priority jobs can be accepted. This can be expected because of the β value which is set to 10. The rule-based method attempts to keep the ratio $P_{Low}/P_{High} = 10$, therefore $P_{High} > 10\%$ leads in $P_{Low} = 1$ which means the system is not accepting any low priority job. The fuzzy-Q learning method also converges to very low C_{Shared} values, but the fuzzy-Q learning method's policy differs from that one of the rule-based. The fuzzy-Q learning method's objective is not to retain the relative Blocking probabilities equal to 10, it attempts to minimize a long term discounted cumulative Blocking. Thus, the C_{Shared} is not totally set to zero.
As mentioned before, the capacity failure scenarios divide the whole simulation duration in to three intervals. In the first interval, the system works normally with default system settings. The second interval starts when the failure in the C_{Total} happens and ends when the failure is repaired. In the third interval the system C_{Total} is brought back to the normal working situation with full capacity. The quality matrix for each of these intervals is listed below as three tables (table 4-5, 4-6 and 4-7).

Simulation scenarios	**Converged cumulative Blocking value**	**Convergence time (Seconds)**	**Time fraction of meeting P_{Low} and P_{High} target**	**90% of P_{High} after convergence**	**90% of P_{Low} after convergence**
5 channels failure, Rule-based method	0.0669866138	83400	0.999673	0.004648	0.036175

5 channels failure, Fuzz-Q Learning method	0.1024201422	147400	0.876452	0.007692	0.10452
10 channels failure, Rule-based method	0.0669866138	83400	0.999673	0.004648	0.036175
10 channels failure, Fuzz-Q Learning method	0.1024201422	147400	0.876452	0.007692	0.10452
20 channels failure, Rule-based method	0.0669866138	83400	0.999673	0.004648	0.036175
20 channels failure, Fuzz-Q Learning method	0.1024201422	147400	0.876452	0.007692	0.10452

Table 4-5: The quality matrix of self-optimization algorithms in the first interval (normal working condition)

Simulation scenarios	**Converged cumulative Blocking value**	**Convergence time (Seconds)**	**Time fraction of meeting P_{Low} and P_{High} target**	**90% of P_{High} after convergence**	**90% of P_{Low} after convergence**
5 channels failure, Rule-based method	0.2520098059	1085100	0.007871	0.016838	0.148068
5 channels failure, Fuzz-Q Learning method	0.2813569193	1149500	0.000706	0.017362	0.26019
10 channels failure, Rule-based method	0.7185525230	1188900.00	0	0.044216	0.389091

10 channels failure, Fuzz-Q Learning method	0.7194092225	1242800.00	0	0.049065	0.40813
20 channels failure, Rule-based method	2.6066953954	1260600	0	0.174102285	1
20 channels failure, Fuzz-Q Learning method	2.6733058054	1418500	0	0.194580965	1

Table 4-6: The quality matrix of self-optimization algorithms in the second interval (failure in the capacity)

Simulation scenarios	**Converged cumulative Blocking value**	**Convergence time (Seconds)**	**Time fraction of meeting P_{Low} and P_{High} target**	**90% of P_{High} after convergence**	**90% of P_{Low} after convergence**
5 channels failure, Rule-based method	0.0561547489	2064200	0.999359	0.004454	0.036769
5 channels failure, Fuzz-Q Learning method	0.1098873154	2122100	0.841763	0.00752	0.122427
10 channels failure, Rule-based method	0.0585011008	2063600	1.00	0.004485	0.036141
10 channels failure, Fuzz-Q Learning method	0.1410742768	2184100	0.833027	0.007626	0.115626
20 channels failure, Rule-based method	0.0755611173	2096700	0.9973	0.004558	0.037067

20 channels failure, Fuzz-Q Learning method	0.1124174657	2147100	0.858386	0.007892	0.138756

Table 4-7: The quality matrix of self-optimization algorithms in the third interval (repairing the failure and back to normal working condition)

The above results show that both algorithms have successfully adapted to the failure and have changed the optimal C_{Shared} value according to the new C_{Total}. I have created the following figures (Figure 4-18 to 4-20) to provide a better visualization of the quality matrix values and to compare both algorithms' performances. Converged cumulative Blocking values, convergence times and time fraction of meeting targeted Blocking for all the scenarios are plotted as below (90% of P_{Low} and P_{High} after convergence figures are provided in appendix):

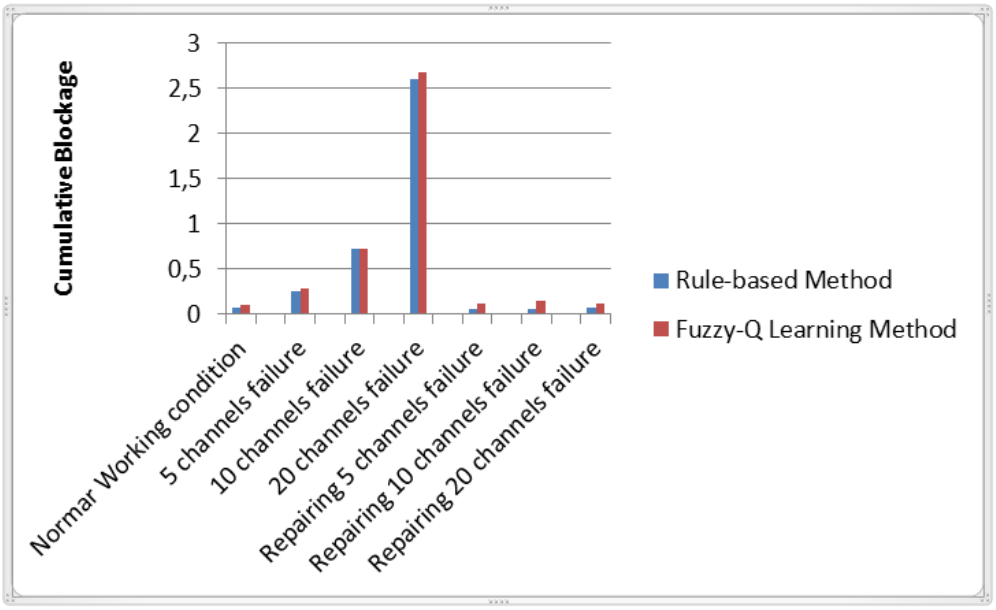


Figure 4.18: Converged cumulative Blockings in different change in capacity scenarios for all the three intervals

In the third interval for 5 and 10 channels failure scenarios, the rule-based method has performed considerably better than the fuzzy-Q learning method and the cumulative Blocking in this interval is even lower than the cumulative reward in the first interval. The explanation is that for the rule-based algorithm, the first and third intervals are basically the same considering that the initialized value for C_{Shared} in the first interval is 10 channels while in the third interval it is higher (28 for the scenario of failure by 5 channels and 22 in the case of happening a 10 channels failure in total capacity) and the convergence from a higher initialized value is less costly. For the fuzzy-Q learning method, facing a new system state plays more important role than the C_{Shared} initialization, therefore the same improvement is not observed for the fuzzy Q-learning method.

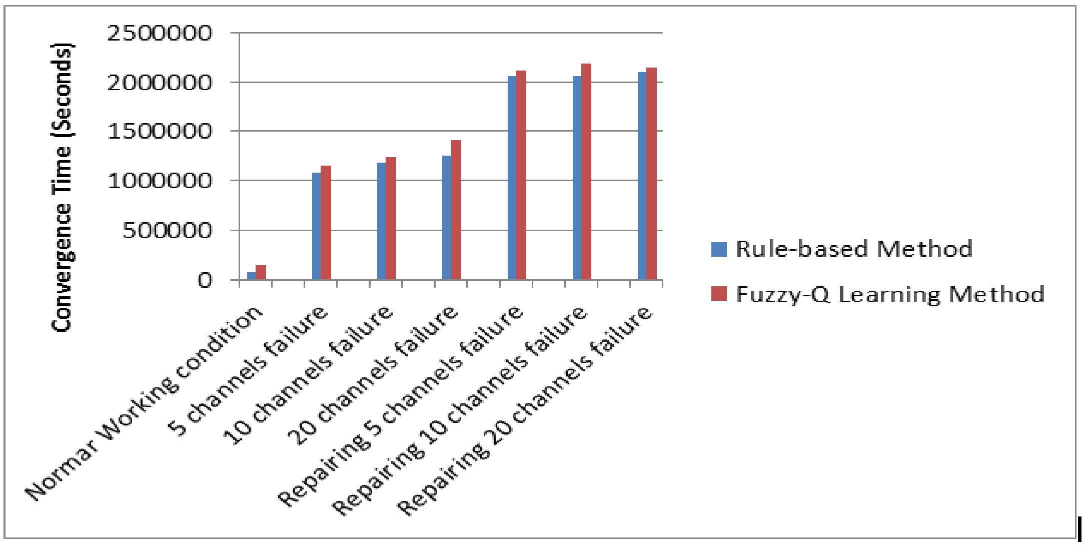


Figure 4.19: Convergence time of cumulative Blockings in different failure in capacity scenarios for all the three intervals

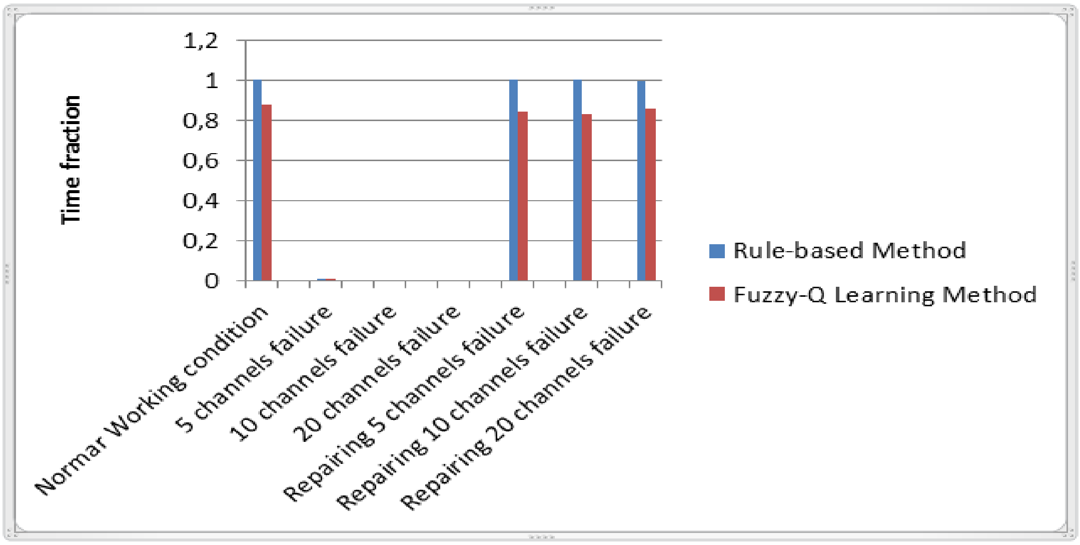


Figure 4.20: Time fraction of meeting targeted Blocking in different change in capacity scenarios for all the three intervals

Figure 4-20 illustrates that in the case of failure in capacity, both of the algorithms were not successful meeting the targeted Blocking (0% of the time). However, after repairing the failure both of the algorithms successfully returns to normal working circumstances and the time fraction of meeting targeted Blocking before and after the failure is relatively the same.

4.3 Change in Arrival Process Scenarios

In the third and final series of scenarios, I will investigate the performance of the self-optimization algorithms in the case of change in arrival process characteristics using batch arrival approach. In the default arrival process, jobs are generated according to the Poisson process and each arrival is equivalent of one arriving job. In the batch arrival, jobs are generated according to the Geometric distribution and each arrival represents the arrival of a batch of jobs. In other words, a group of jobs called batch arrives and the average size of the batch is one of the arrival process parameters which needs to be set.

In the sections below, I will define two general sets of simulation scenarios. In the first set of simulations, at t= 1000000, the arrival process changes gradually over an hour (3600 seconds) to a batch arrival process with different average batch sizes. As the number of expected arrivals per time (λ) is constant, this batch arrival process means that not only will the system have to deal with multiple jobs at the moment of arrival, but it must also deal with an increased number of calls. For example, increasing the batch size to two or three while the λ is constant means that the total number of arrivals will be two or three times higher than before.

In the second set of scenarios, while increasing the batch size, the λ is decreased by the same rate. For example if in one hour the arrival process has changed to a Geometric distribution with batch size of two, then, the λ will be decreased from 30 to 15. In this set of simulation scenarios, the total number of arrivals will not be increased, but instead of jobs arriving one by one they arrive in a different average batch of sizes.

All the jobs in one batch arrive simultaneously, but each job has independent duration according to the Poisson process with the mean value of *1/μ*. Furthermore, jobs are accepted and processes in the system one by one which means that the system does not deal all the jobs in one batch as "one job which requires more than one channel " or "one job with longer processing time".

4.3.1 Batch Arrival Scenarios without Changing the Arrival Rate (λ)

Performance results of changing the arrival process to batch arrival with batch size two and without decreasing the λ, are recorded in Table 4-8.

Simulation scenarios	**Converged cumulative Blocking value**	**Convergence time (Seconds)**	**Time fraction of meeting P_{Low} and P_{High} target**	**90% of P_{High} after convergence**	**90% of P_{Low} after convergence**
Rule-based method, Batch size 2	1.6137291539	1264700	0	0.089520534	0.837869823
Fuzzy-Q learning method, Batch size 2	2.082705434	1062000	0	0.108465369	1
Rule-based method, Batch size 3	3.0775641267	1291900	0	0.221235691	1

Fuzzy-Q learning method, Batch size 3	4.4229599011	1058200	0	0.332876059	1

Table 4-8: Performance quality matrix of algorithms for different scenarios of change in arrival batch size

It is illustrated by Table 4-8 values that increasing the batch size without decreasing λ leads to a dramatic rise in the number of arrivals and, consequently, P_{Low} and P_{High}. The self-optimization algorithms' solution towards this increase in arrivals is blocking the system for low priority jobs and only accepting high priority jobs. To provide more visualization of the quality matrix values, I displayed the converged cumulative Blocking values and the convergence time in Figures 4-21 and 4-22. The time fraction of meeting the targeted Blocking for all of the scenarios is zero so I have not include the coordinated figure (90% of P_{Low} and P_{High} values are provided in the Appendix).

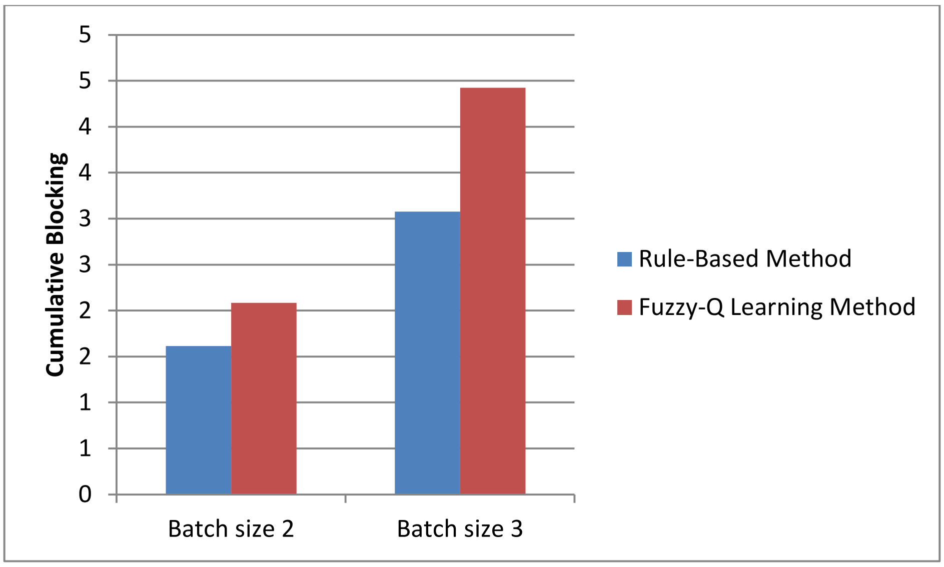


Figure 4.21: The converged cumulative Blocking for different batch arrival scenarios

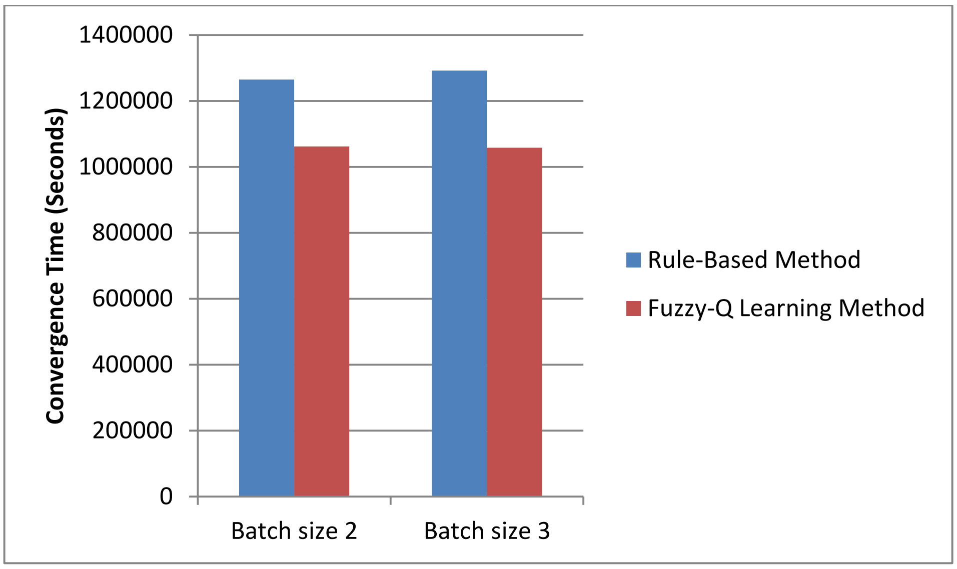


Figure 4.22: The convergence time of cumulative Blocking for different batch arrival scenarios

Comparing the rule-based method to the fuzzy-Q learning method performances, it is evident that the latter's cumulative Blockings converges faster and the converged values are higher than the formers.

4.3.2 Batch Arrival Scenarios with Changing the Arrival Rate (λ)

The second set of simulations is increasing batch size while decreasing λ. In this part of simulation for further investigation, average batch arrival size four is added to the scenarios to observe the algorithm's reaction to the higher batch sizes. In this set of scenarios, by increasing the batch size to two, three and four, the λ is decreased to λ/2, λ/3 and λ/4. I have presented the algorithms' performance towards changes in arrival process in Figures 4-23 to 4-28.

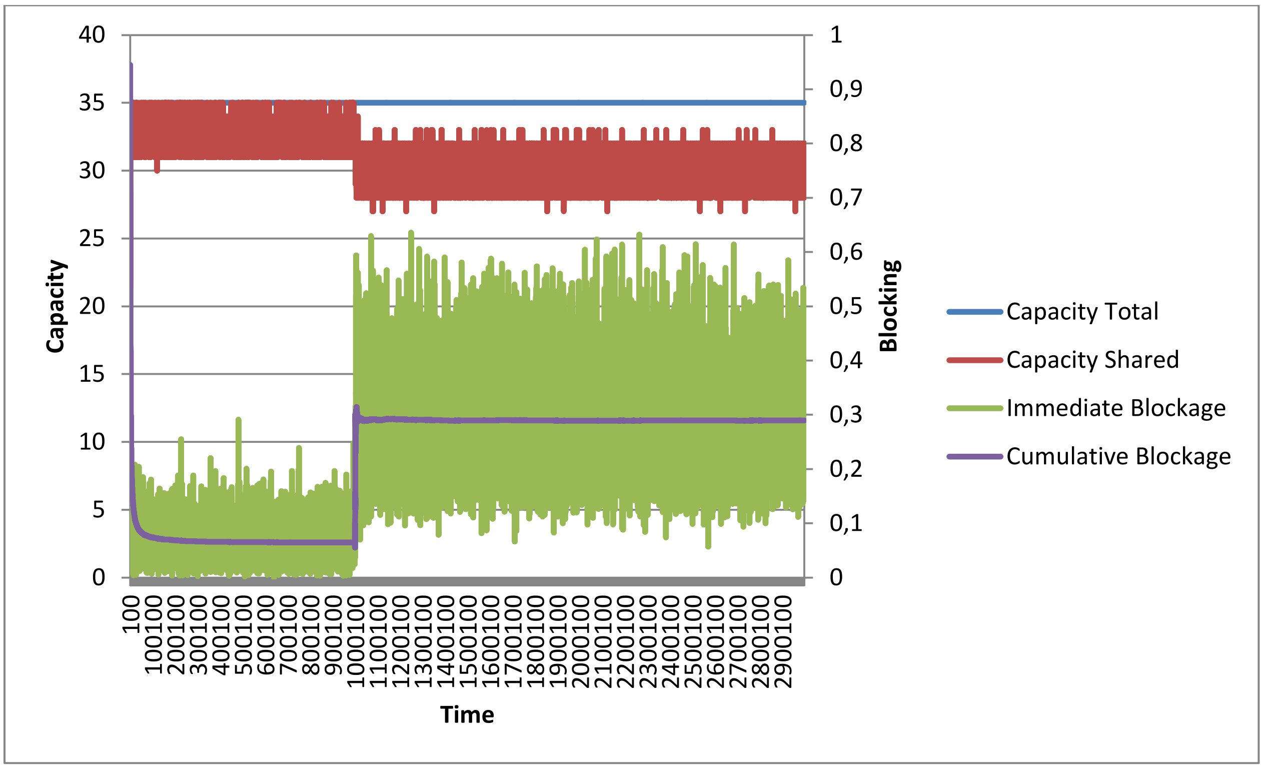


Figure 4.23: Rule-based method's reaction to the batch arrival (average batch size=2)

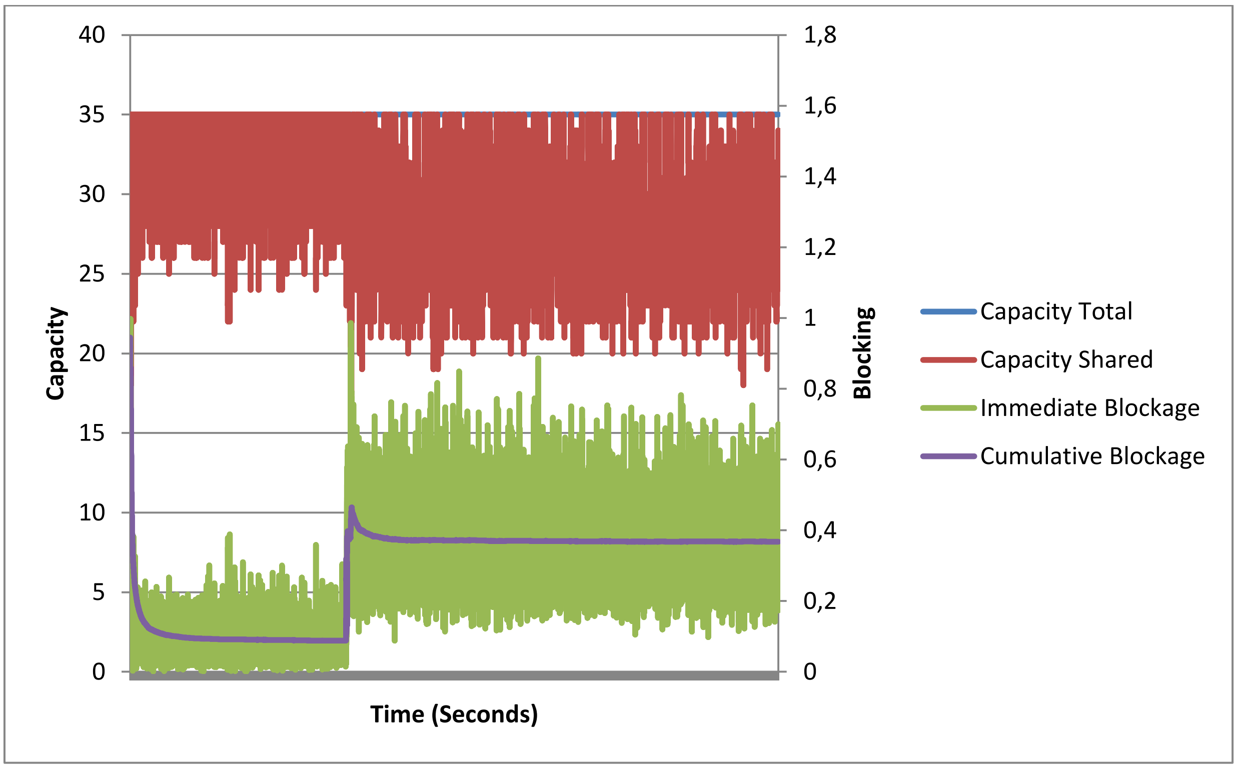


Figure 4.24: Fuzzy-Q learning method's reaction to the batch arrival (average batch size=2)

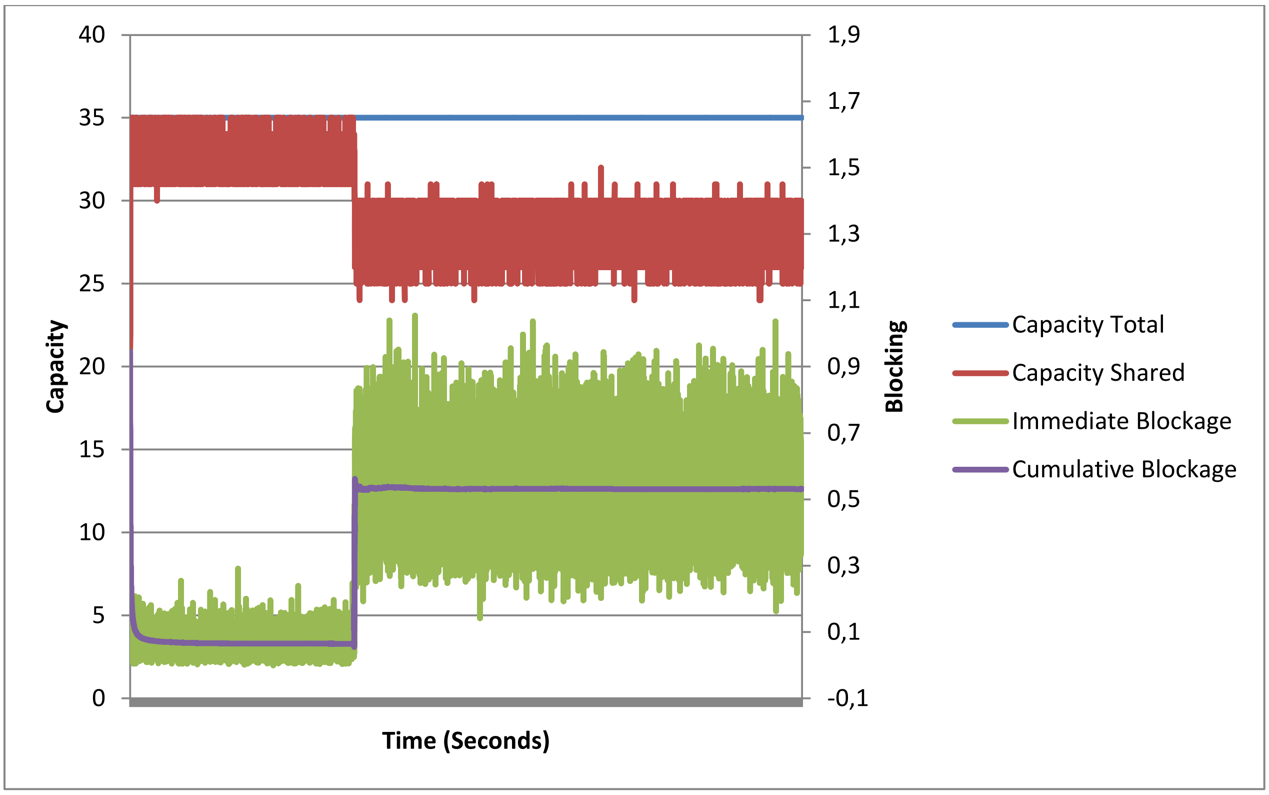


Figure 4.25: Rule-based method's reaction to the batch arrival (average batch size=3)

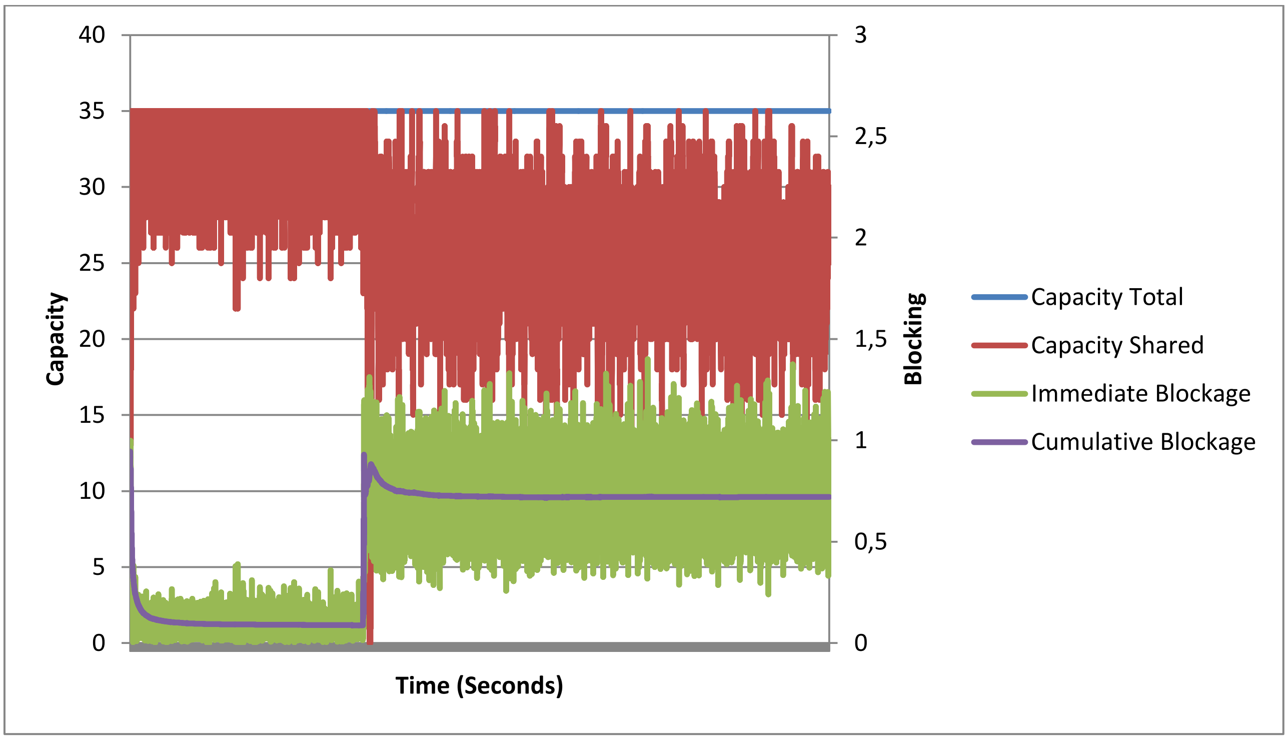


Figure 4.26: Fuzzy-Q learning method’s reaction to the batch arrival (average batch size=3)

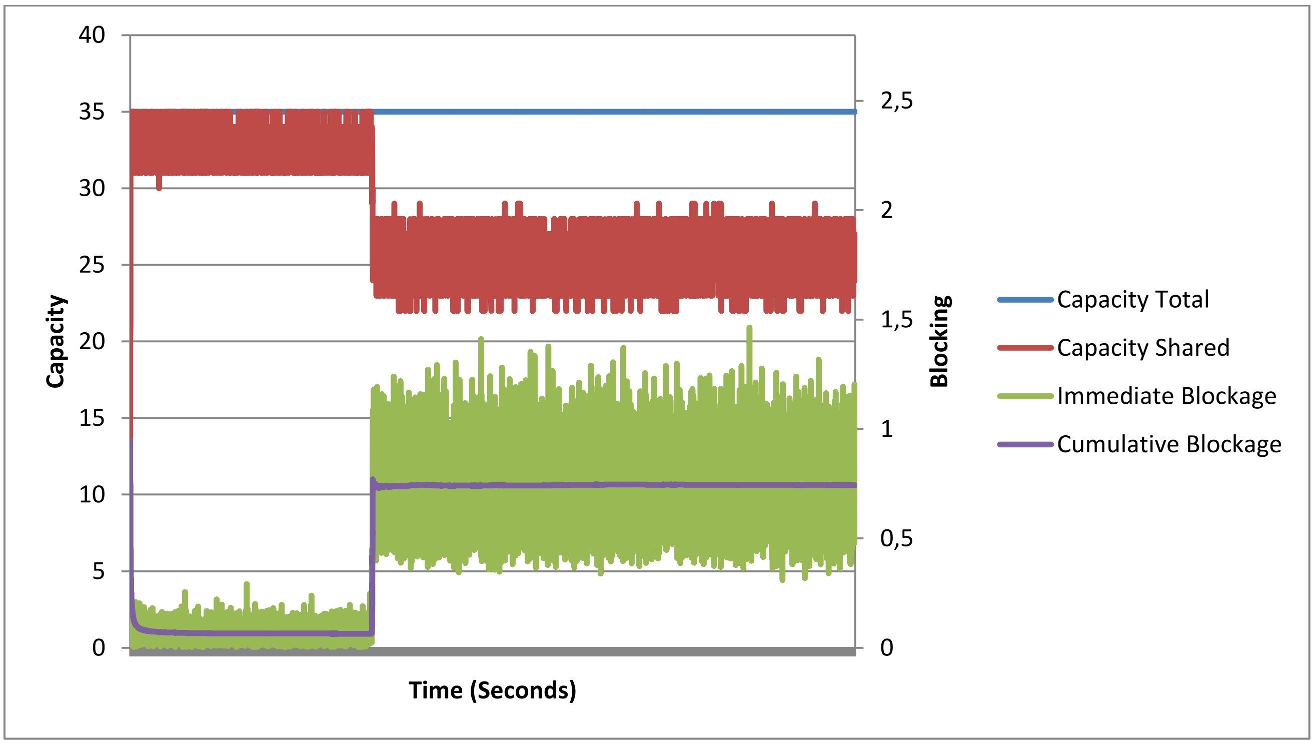


Figure 4.27: Rule-based method’s reaction to the batch arrival (average batch size=4)

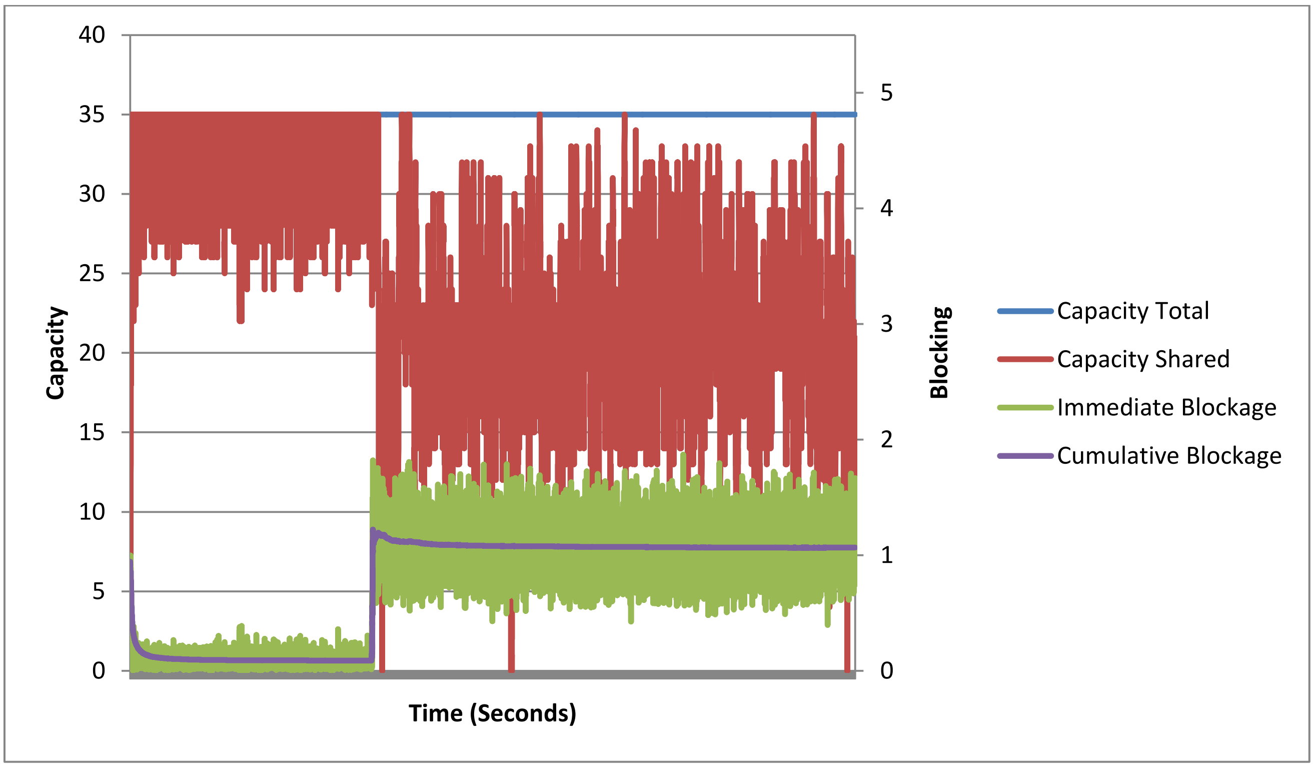


Figure 4.28: Fuzzy-Q learning method's reaction to the batch arrival (average batch size=4)

By observing above figures it is possible to conclude that although increasing batch size and decreasing λ does not change the total number of arrivals in a time unit, it does increase P_{Low} and P_{High} according to the batch size. This growth in blocking probabilities can be explained by the fact that by changing the batch size the jobs arrive in a batch whereas the departure process of jobs does not change and jobs leave the capacity one by one after job durations. This disagreement in arriving and departing jobs increase the blocking probabilities. The higher the blocking probabilities are the more reserved capacity for high priority jobs is needed in order to provide higher QoS for high priority jobs. It is also important to take note of the high fluctuation in immediate reward values. The batch arrival affects the Blocking in a dramatic way because of its highly dynamic nature.
Table 4-9 contains the quality matrix values of algorithms for all the scenarios.

Simulation scenarios	Converged cumulative Blocking value	Convergence time (Seconds)	Time fraction of meeting P_{Low} and P_{High} target	90% of P_{High} after convergence	90% of P_{Low} after convergence
Rule-based method, Batch size 2	0.2921340385	1159100	0.006	0.019352505	0.17171176

Fuzzy-Q learning method, Batch size 2	0.3743656398	1212900	0	0.026871757	0.313081555
Rule-based method, Batch size 3	0.5332015231	1267300	0	0.034000354	0.302417236
Fuzzy-Q learning method, Batch size 3	0.7996475670	1025100	0	0.05331669	0.524251259
Rule-based method, Batch size 4	0.7408968521	1286100	0	0.046799355	0.417725014
Fuzzy-Q learning method, Batch size 4	1.0888432272	1297500	0	0.072972973	0.72802042

Table 4-9: Performance quality matrix of algorithms for different scenarios of simultaneous change in λ and batch arrival size

The quality of the performance matrix's values represent that although the convergence time of the cumulative Blocking for both of the algorithms is relatively similar, the converged cumulative Blocking of the fuzzy-Q learning method is higher compared to the rule-based method. The reason is that the strong fluctuation of immediate reward, resulted by the batch arrival process, affects the fuzzy-Q learning method's performance more than the rule-based method because the fuzzy-Q learning method is more sensitive to change in the system state. A minor change in the state after fuzzification might result in totally different sub-states. The fuzzy-Q learning method decision making logic is totally dependent on previous experiences and learning from former action performances. Thus a stable system is preferable with this method in order to make a precise assessment about the former taken actions' performance. A dynamic arrival process with fluctuating blocking probabilities may influence the algorithms' assessment in a negative way. For example, a taken action at time *t* can be considered to be an inappropriate one because the Blocking at *t+Δt* is higher than before. Therefore, the action would be punished and not repeated in the same sub-state again while the action could be a proper one and the higher blocking probabilities are the result of dynamic change in the arrival process. To compare the method's performance easily I have provided the converged value of cumulative Blocking and convergence time values of both algorithms in all the scenarios in Figure 4-29 and 4-230.

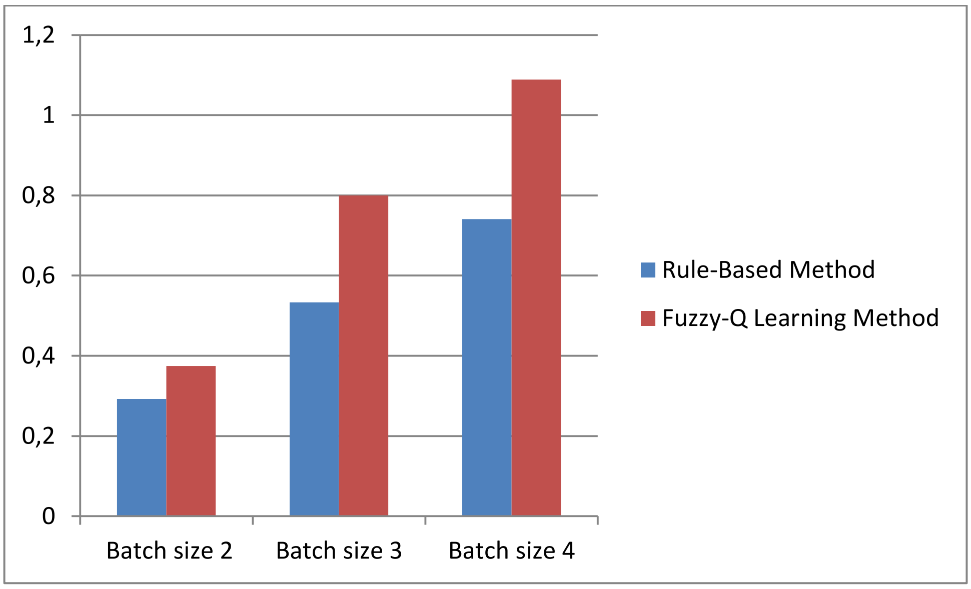


Figure 4.29: Converged cumulative Blocking of algorithms in batch arrival process with changing λ

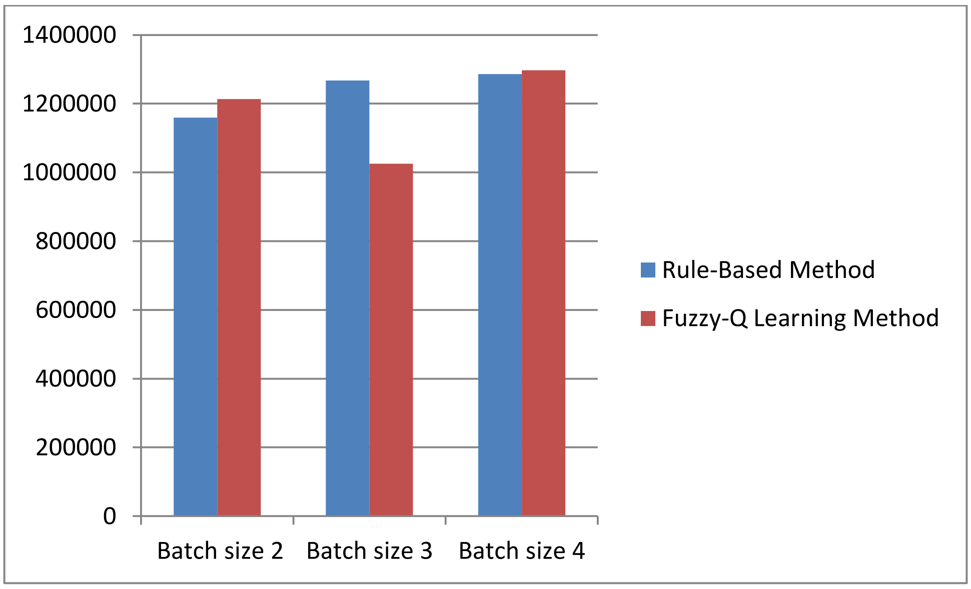


Figure 4.30: Convergence time of cumulative Blocking of algorithms in batch arrival process with changing λ

5 Conclusion and Future Work

In this thesis work I have investigated a self-optimization approach of resource allocation in ICT systems. The case study that I have designed for this research is a system with a total capacity of C_{Total} serving two classes of jobs: low and high priority. Rejecting a high priority job is more costly for the service provider and, as a result, the system must designate some of the resources solely for high priority jobs. This allotment of resources allows the system to comply with various standards of QoS. The objective of the self-optimization methodologies is to minimize Blockings while splitting the C_{Total} into C_{Shared} and $C_{Reserved}$.

I used two different methods to apply Self-organization algorithms to the controller agent of the system, two different methods have been chosen. The first is the rule-based method which is a set of if-then rules written by a knowledgeable human expert, adapting C_{Shared} value according to the system state, reported at the end of each observation interval. The second method is the fuzzy-Q learning method which is one of the reinforcement learning algorithms. The fuzzy-Q learning algorithm does not have any prior knowledge about the system state or the optimal action for each state, but the controller agent is able to learn from former experiences by executing an action and then assessing the success of that action based on the feedback from the system.

I have designed three sets simulations to observe and compare the methods performances. The first set of scenarios' objective is to analyze the effect of different algorithms' parameter settings on the overall performance of the methods. The second set of experiments' was simulating an abrupt failure in capacity and to then study how the algorithms react and adapt to this change. The last set of scenarios simulated a change in the arrival process, shifting from the Poisson arrival process to the multi-sized Geometric distribution batch arrival process, In one group the λ was unchanged while in the another group , it was decreased by the same rate of growth in batch size.

The overall results show that the rule-based method performs better than the Fuzzy-Q Learning method, which is not surprising because the case study that I employed allows the algorithms to easily predict which action is best in each state based on the reported P_{Low} (t) and P_{High} (t). However, considering the cost of the learning and exploration phases of the Fuzzy-Q Learning method, this method also performed acceptably in most of the scenarios.

The first set of simulations indicates that the most influential parameter on the algorithms' overall performance is the size of the observation time interval. The short observation time interval does not allow for the reporting of a reliable state because there is an insufficient amount of arrivals have been observed. This unreliable state report leads to deterioration of the algorithms' performances. Conversely, longer observation time intervals do ensure that the system state has been reported accurately to the controller agent, but the system reacts more slowly and convergence is delayed. It was also highlighted by the simulations' results that the length of observation time should be chosen according to the arrival rate λ, regardless of the applied self-optimization algorithms.
Another important parameter, influencing the fuzzy-Q learning algorithm's performance is the experimenting rate which indicates how often the controller agent chooses a random action to explore (exploration phase).

The failure in capacity simulation results illustrate that, in the case of a failure in capacity, the C_{Shared} is set to a lower value as the consequence of increase in blocking probabilities. If the failure occurs on a larger scale, the system is blocked for low priority jobs altogether and only services high priority jobs. Another noteworthy observation is that when a larger failure takes place, algorithms need more time to adapt and to find the new optimal C_{Shared} compare to a minor failure. However, it is also important to note that after repairing the failure, algorithms can successfully return the system to a normal state after a reasonable time which is promising.

The last set of scenarios demonstrate that the larger size of the arrival batch, even when the λ is decreased by the same rate, results in higher blocking probabilities and consequently a lower C_{Shared}. It is also evident that the fuzzy-Q learning method performs worse than the rule-based method in all of the batch arrival process scenarios. This is due to the fact that fluctuation in the immediate Blocking affects the fuzzy-Q learning method's performance much more than the performance of the rule-based method.

One direction of future research for scholars can be designing case studies in which learning self-optimization methods like the Fuzzy-Q Learning method performs more efficiently than rule-based methods. The case study that I considered for this research, with its direct relationship between blocking probabilities and the optimal C_{Shared} value, allowed for the easy prediction of the most appropriate action in different states of the system. Therefore a knowledgeable expert can set up the look-up table in the way that the controller agent does not experience any difficulties finding the best possible action in each state. But in more complicated case studies it might be not easy (even for a human expert) to predict the best action in the different system states due to unknown and dynamic process affecting the system. For example in a game like chess, the best action and reward feedback in each state is difficult to predict, because the reward is also dependent on the opponent's reaction.
Additionally, scholars could consider and design a case study in which the parameters representing the system state are not easily measureable. For example in my case study two parameters reporting the system state were blocking probabilities which can be measure and reported easily while it is possible that in a cellular mobile network, the blocking probabilities in neighboring cells are matter of importance and for the controller agent it is not easy to measure blocking probabilities in other cells.

Moreover, as an improvement for the fuzzy-Q learning, I suggest an algorithm with variable experimenting rate learning rates. At the beginning of any self-optimization process, the fuzzy-Q learning is in the exploration phase and as the time progresses it shifts to utilization of learnt information more often (exploitation phase). The higher value of experimenting and learning rates accelerate the exploration procedure, but it is not preferable when the algorithm is in exploitation phase. Therefore, a dynamic experimenting and learning rate which is high at the beginning of the fuzzy-Q learning method's employment and decreasing as time passes improves the algorithm's performance.

Bibliography

[1] Nec. (2009). Self Organizing Network, NEC's proposals for nextgeneration radio network management.

[2] J.L. van den Berg,R.Litjens. (2008). SOCRATES: Self-Optimisation and self-ConfiguRATion in wirelESs networks. *COST 2100 TD(08)422, Wroclaw, Poland*.

[3] Mcauley, A., & Manousakis, K. (2000). SELF-CONFIGURING NETWORKS. *MILCOM 2000. 21st Century Military Communications Conference Proceedings* .

[4] Debanjan.G, S. R. (2007). Self-healing systems — survey and synthesis. *Decision Support Systems 42 (2007) 2164–218*.

[5] Yilmaz, O. (2010). 'Self-Optimization of Coverage and Capacity in LTE using Adaptive Antenna System'. *Master's Thesis, AALTO university <http://lib.tkk.fi/Dipl/2010/urn100152.pdf>*.

[6] Nasri, R. A. (2006). 'Fuzzy-Q-Learning-Based Autonomic Management of Macro-diversity Algorithm in UMTS Networks'. *ANNALES DES TELECOMMUNICATIONS, VOL 61; PART 9/10, p. 1119-1135* .

[7] Bahsoon, R. (2010). 'Green Cloud: Towards a Framework for Dynamic Self-Optimization of Power and Dependability Requirements in Cloud Architectures. *ECSA'10*, 510-514.

[8] Nguyen Van, H. D. (n.d.). 'Autonomic virtual resource management for service hosting platforms'. *Proc. of the Workshop on Software Engineering Challenges in Cloud Computing*, 2009.

[9] Vouk, M. (2008). Cloud computing–Issues, research and implementations. *Journal of Computing and Information Technology - CIT 16*, 235–246.

[10] Michael Armbrust, A. F. (2009). Above the clouds: A berkeley view of cloud computing. *Technical Report No. UCB/EECS-2009-28*.

[11] Qing-An Zeng, D. P. (2002). Handoff in wireless mobile networks. *Handbook of wireless networks and mobile computing, John Wiley & Sons, Inc., New York, NY, USA*, 1-25.

[12] Crina Grosan, A. A. (2011). *Intelligent Systems: A Modern Approach.* 149-153: Intelligent Systems Reference Library Series, Springer Verlag, Germany.

[13] Abraham, A. (2005). 130: Rule-based Expert Systems. *Handbook of Measuring System Design, edited by Peter H. Sydenham and Richard Thorn, John Wiley & Sons, Ltd. ISBN: 0-470-02143-8*, 909-919.

[14] Adedeji Bodunde Badiru, P. M.-B. (2002). Fuzzy engineering expert systems with neural network applications. John Wiley & Sons, Inc. New York, NY, USA.

[15] Peter Dayan, C. J. (2001). *Reinforcement Learning.* Encyclopedia of Cognitive Science London, England: MacMillan Press.

[16] Tom Mitchell, M. H. (1997). *Machine Learning.* 367-390: McGraw-Hill Science Engineering.

[17], [19] Richard S. Sutton, A. G. (1998). *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, A Bradford Book.

[18], [20] Leslie Pack Kaelbling, M. L. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 237-285.

[21] Smith, M. (2006). *Markov Decision Processes & Reinforcement Learning.* Lehigh University.

[22] Hellmann, M. (2001). *Fuzzy logic introductio.* Laboratoire Antennes Radar Telecom.

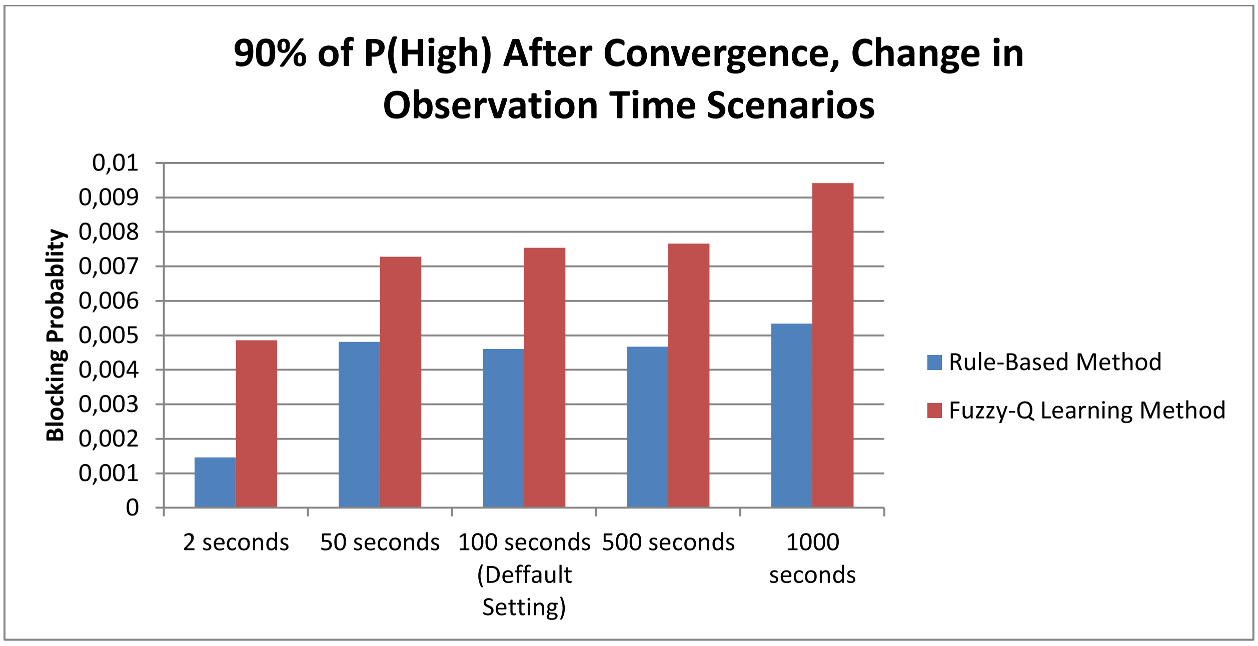
[23] Watkins, C. J. (1989). *Learning from delayed rewards.* PhD Thesis, University of Cambridge, England.

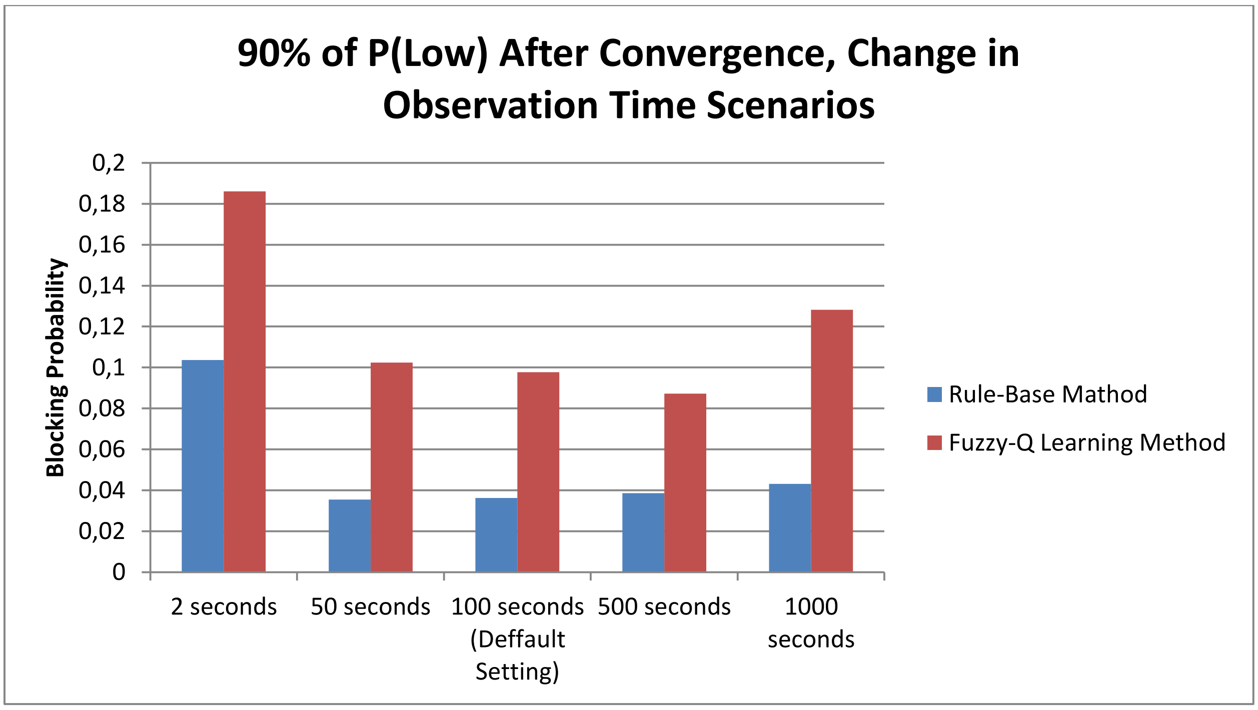
[24] Eder, C. (2008). *Q-Learning:A Simple Reinforcement Learning Algorithm Based On The Temporal Difference Approach.* Version 18. Knol.

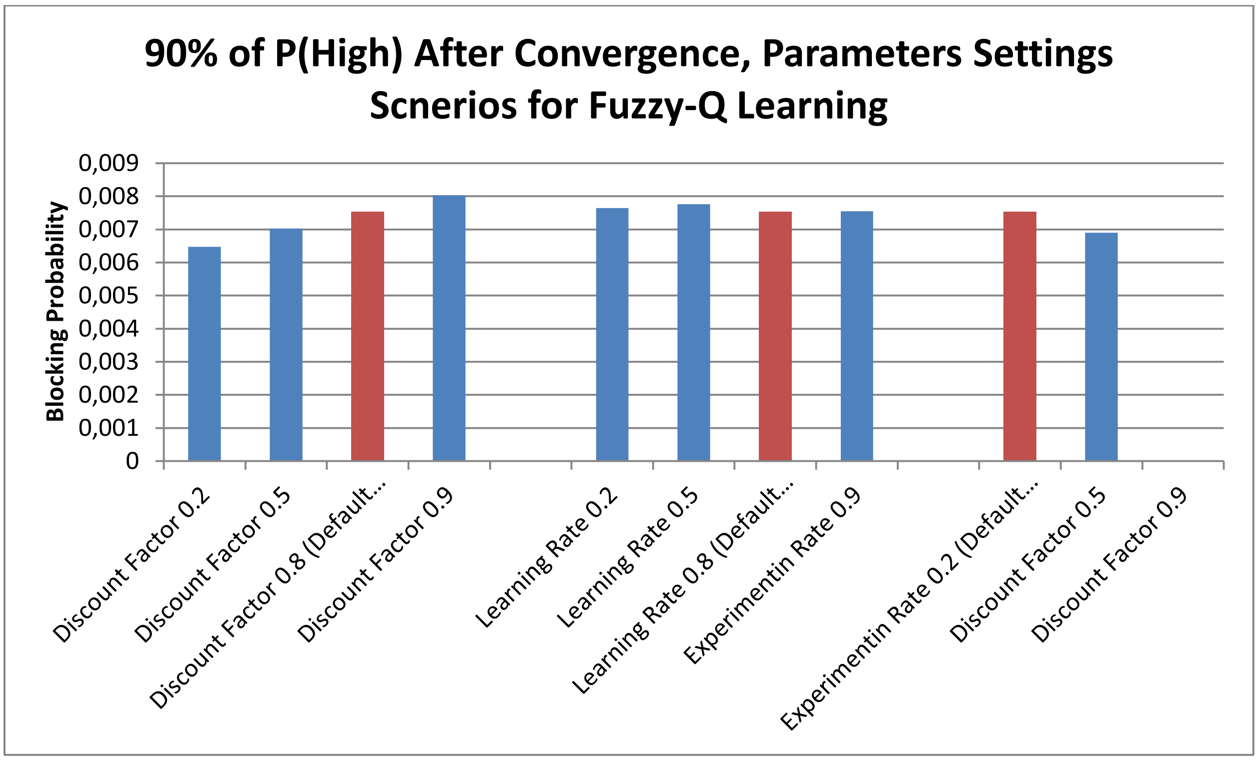
[25] by Eyal Even-dar, Y. M. (2003). Learning Rates for Q-learning. *Journal of Machine Learning Research*, 1-25.

Appendix A

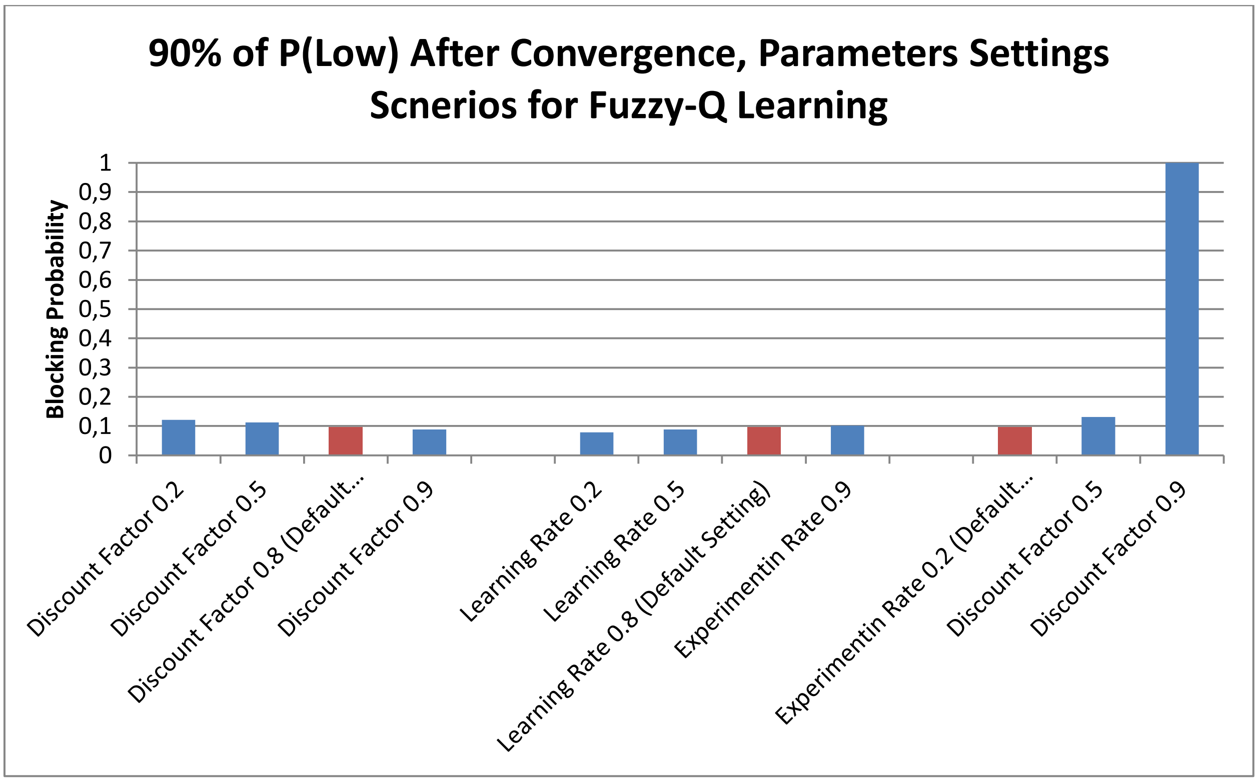
90% of P_{Low} and P_{High} Figures for differens simulation scenarios.



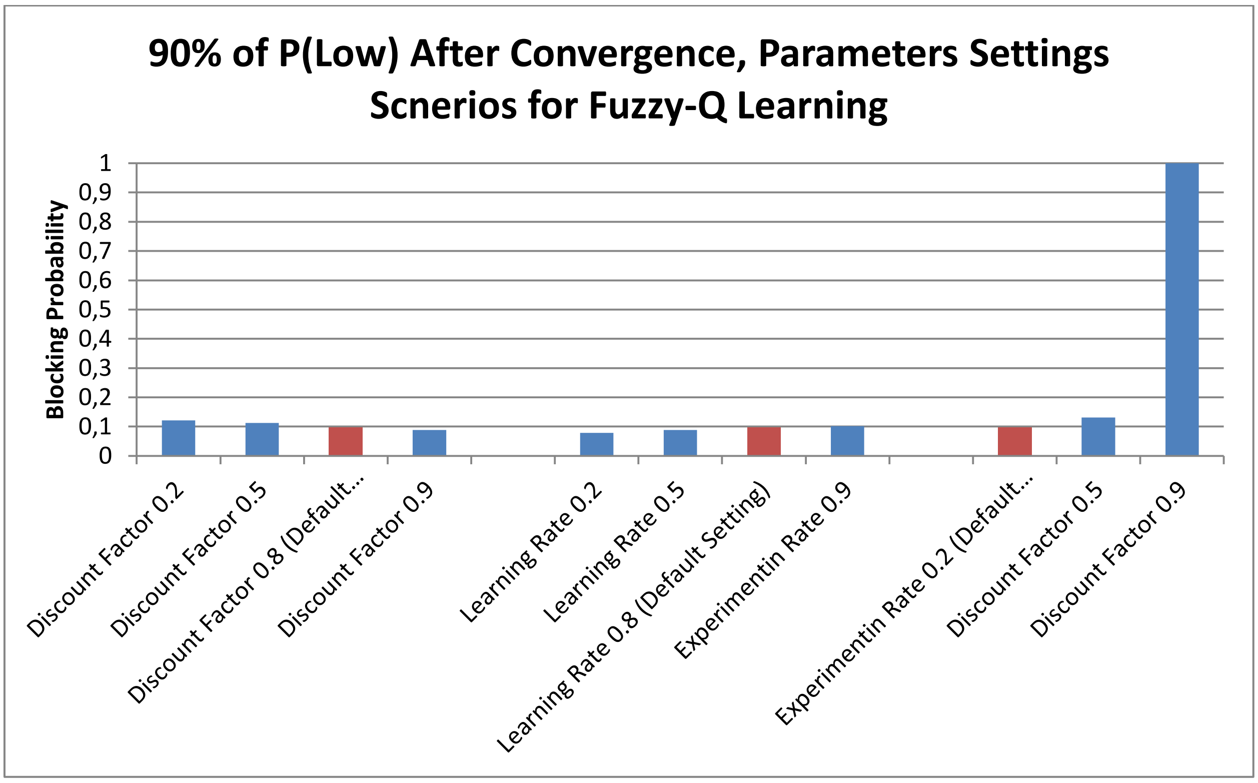


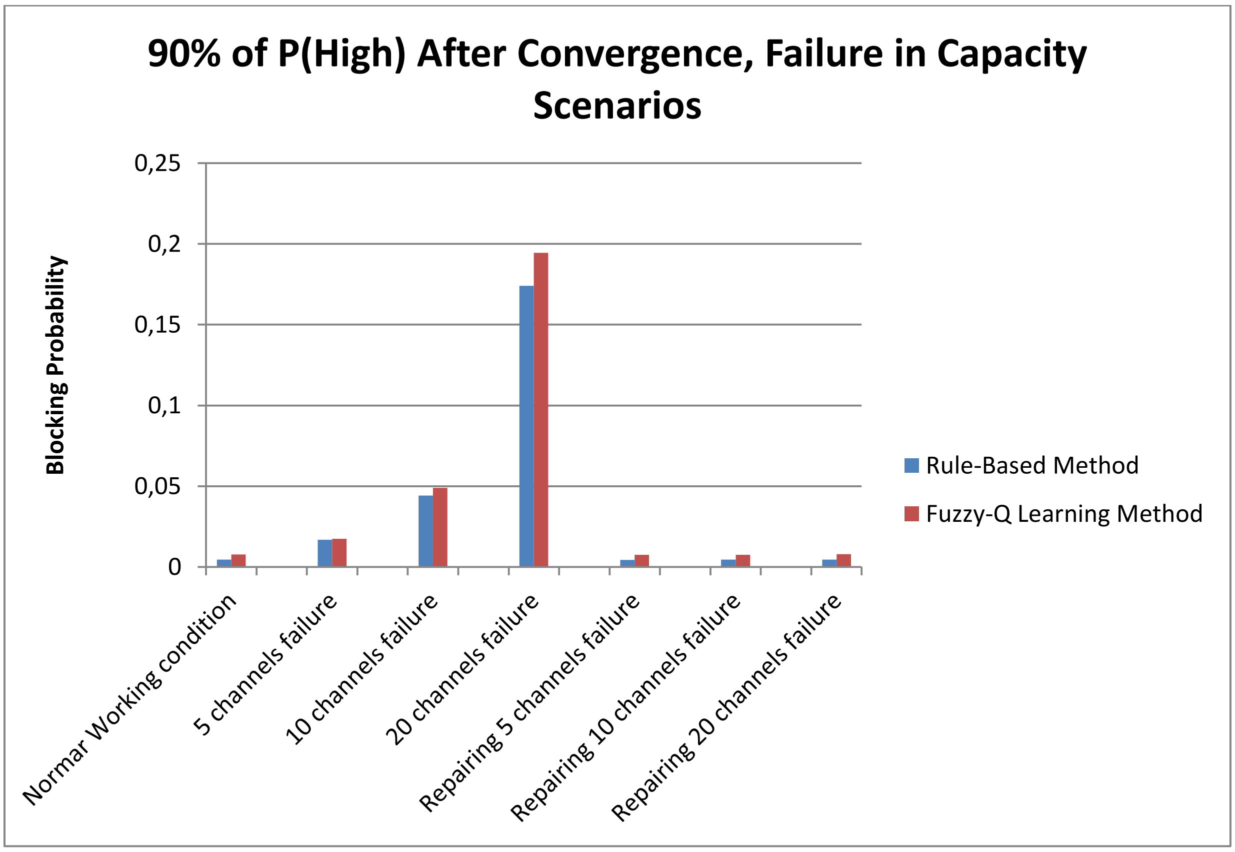


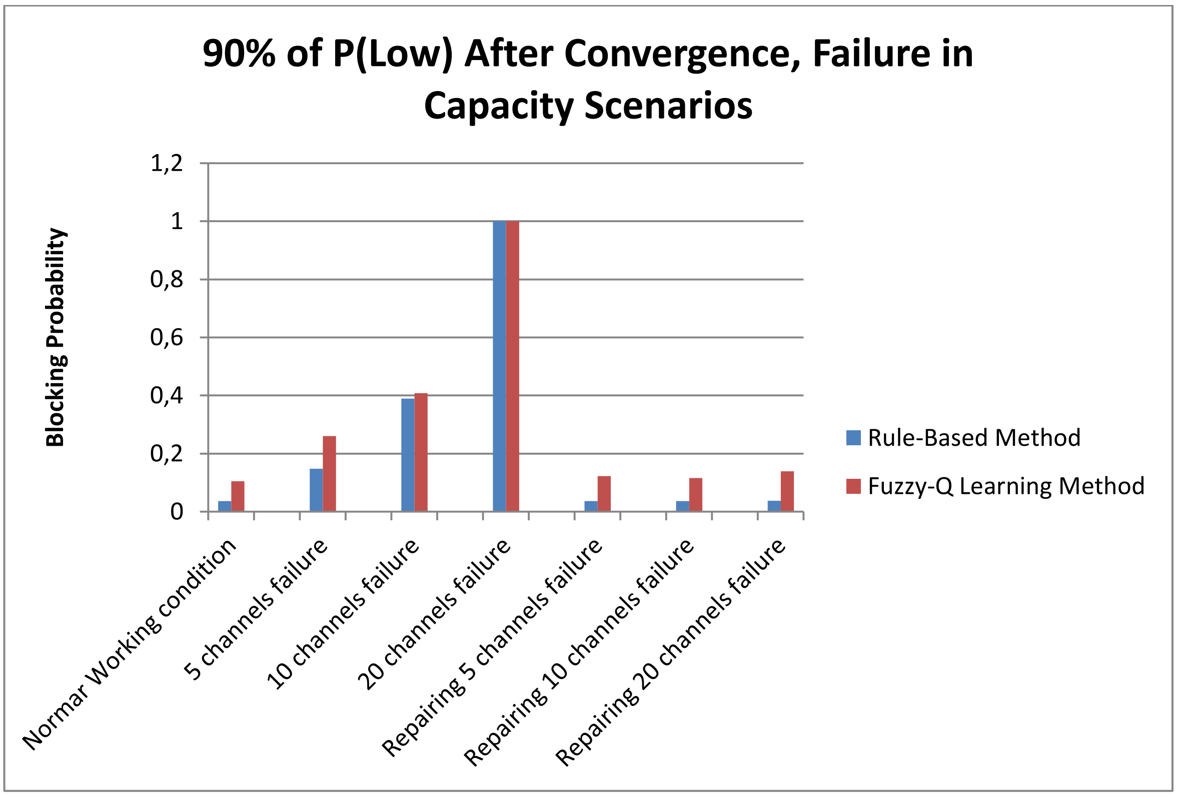
90% of P(High) After Convergence, Parameters Settings Scnerios for Fuzzy-Q Learning
Blocking Probability
0,009
0,008
0,007
0,006
0,005
0,004
0,003
0,002
0,001
0
Discount Factor 0.2
Discount Factor 0.5
Discount Factor 0.8 (Default…
Discount Factor 0.9
Learning Rate 0.2
Learning Rate 0.5
Learning Rate 0.8 (Default…
Experimentin Rate 0.9
Experimentin Rate 0.2 (Default…
Discount Factor 0.5
Discount Factor 0.9

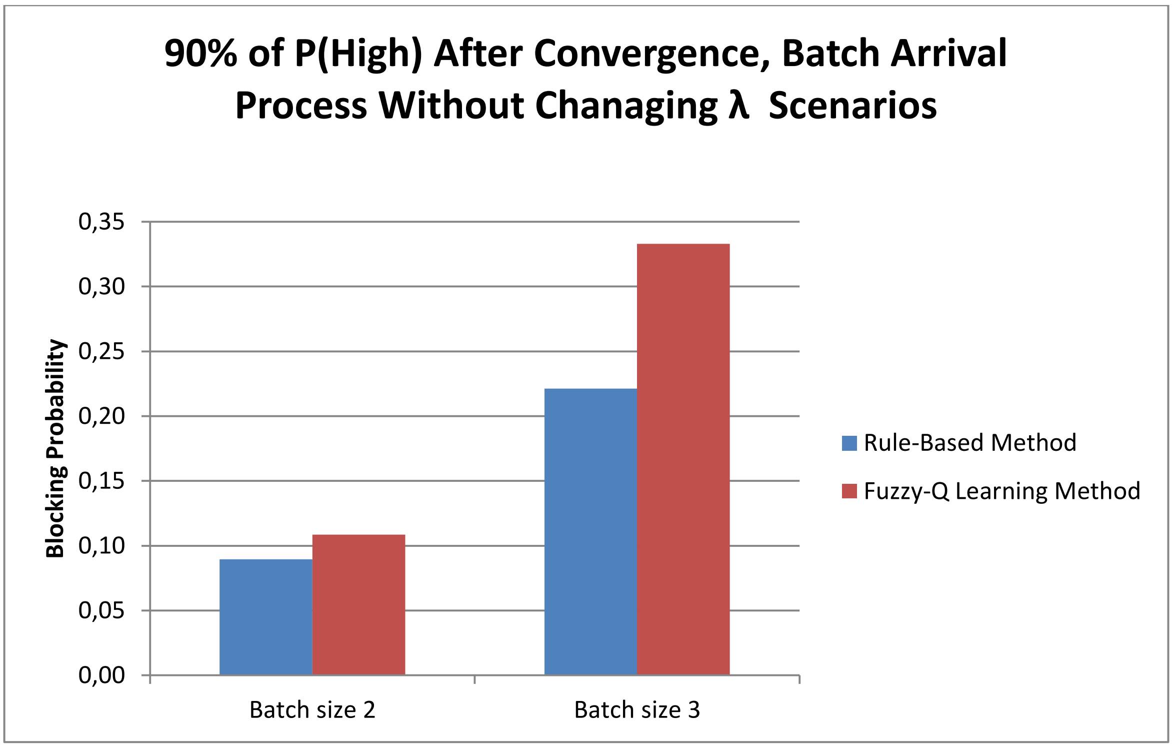


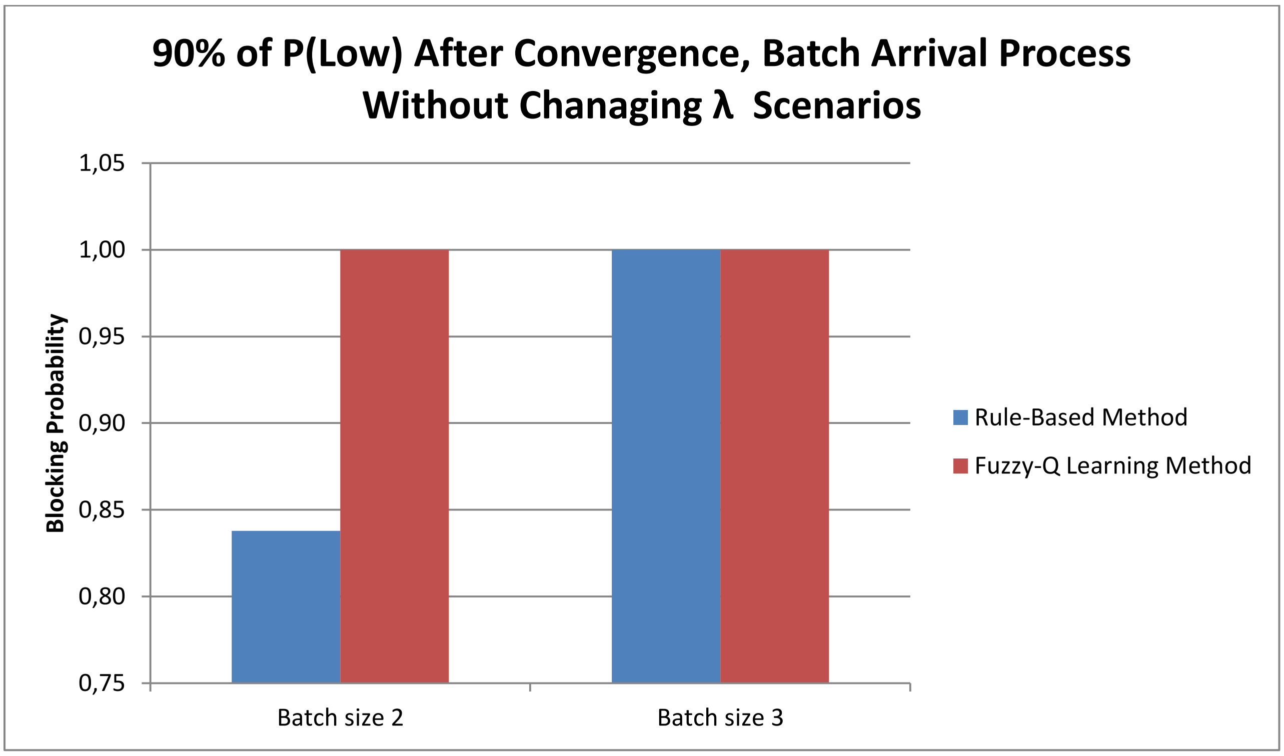
90% of P(Low) After Convergence, Parameters Settings Scnerios for Fuzzy-Q Learning
Blocking Probability
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
Discount Factor 0.2
Discount Factor 0.5
Discount Factor 0.8 (Default…
Discount Factor 0.9
Learning Rate 0.2
Learning Rate 0.5
Learning Rate 0.8 (Default Setting)
Experimentin Rate 0.9
Experimentin Rate 0.2 (Default…
Discount Factor 0.5
Discount Factor 0.9

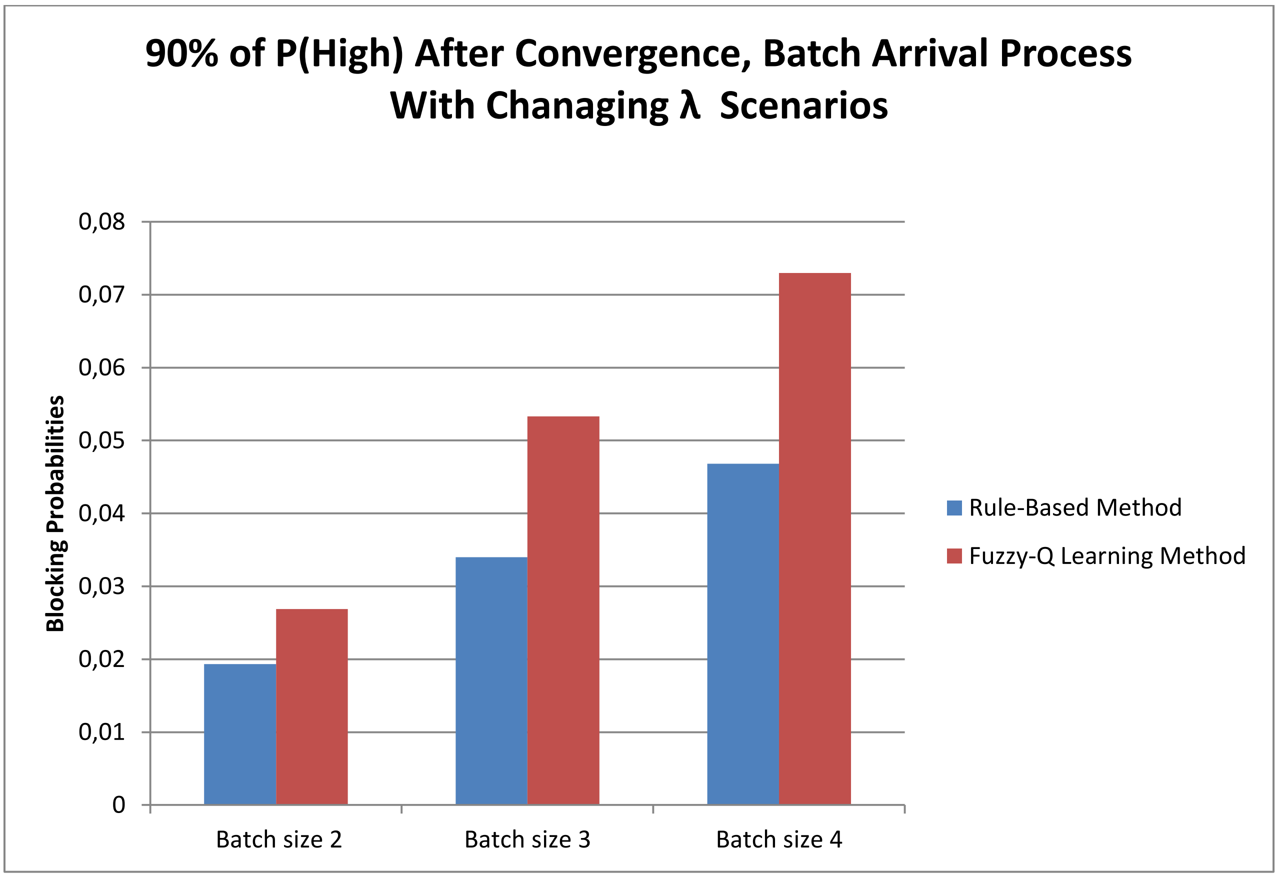

90% of P(Low) After Convergence, Parameters Settings Scnerios for Fuzzy-Q Learning
Blocking Probability
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
Discount Factor 0.2
Discount Factor 0.5
Discount Factor 0.8 (Default…
Discount Factor 0.9
Learning Rate 0.2
Learning Rate 0.5
Learning Rate 0.8 (Default Setting)
Experimentin Rate 0.9
Experimentin Rate 0.2 (Default…
Discount Factor 0.5
Discount Factor 0.9

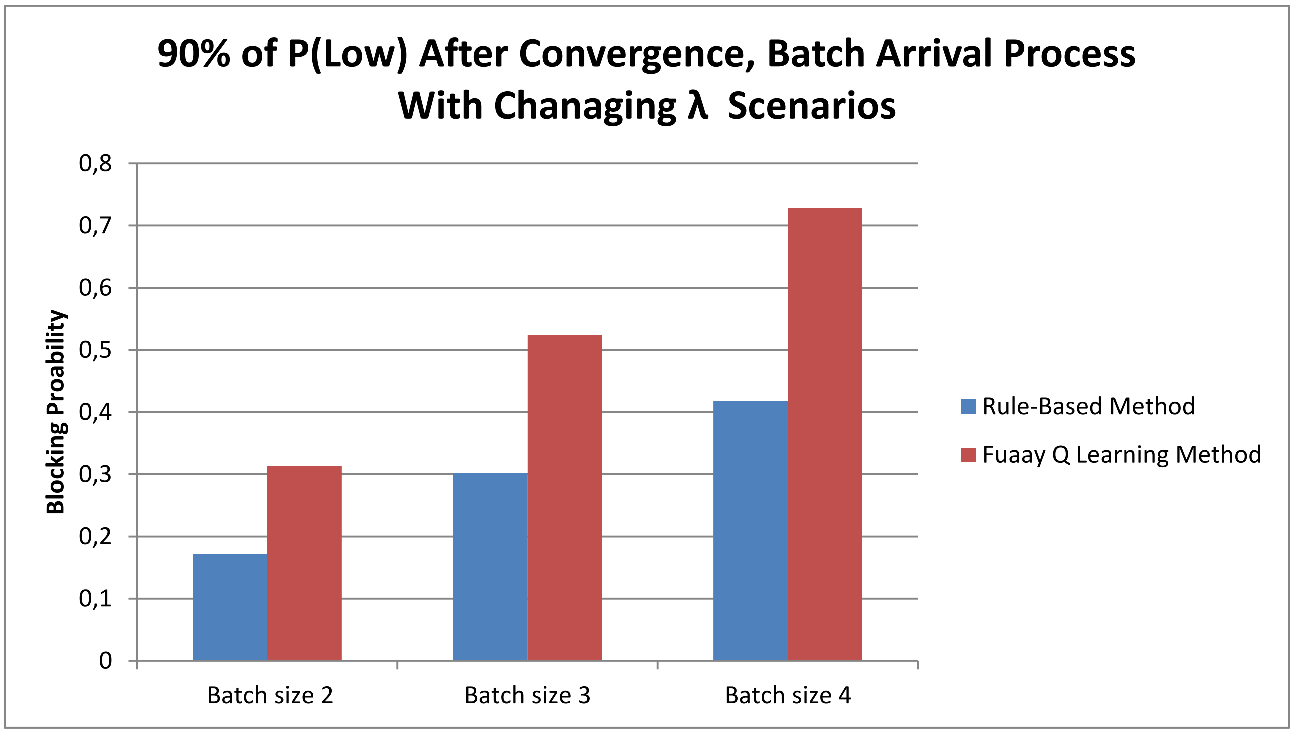

90% of P(High) After Convergence, Failure in Capacity Scenarios
Blocking Probability
0,25
0,2
0,15
0,1
0,05
0
Normar Working condition
5 channels failure
10 channels failure
20 channels failure
Repairing 5 channels failure
Repairing 10 channels failure
Repairing 20 channels failure
🟦 Rule-Based Method
🟥 Fuzzy-Q Learning Method


90% of P(Low) After Convergence, Failure in Capacity Scenarios
Blocking Probability
1,2
1
0,8
0,6
0,4
0,2
0
Normar Working condition
5 channels failure
10 channels failure
20 channels failure
Repairing 5 channels failure
Repairing 10 channels failure
Repairing 20 channels failure
Rule-Based Method
Fuzzy-Q Learning Method


90% of P(High) After Convergence, Batch Arrival Process Without Chanaging λ Scenarios
Blocking Probability
0,35
0,30
0,25
0,20
0,15
0,10
0,05
0,00
Batch size 2
Batch size 3
Rule-Based Method
Fuzzy-Q Learning Method


90% of P(Low) After Convergence, Batch Arrival Process Without Chanaging λ Scenarios
Blocking Probability
1,05
1,00
0,95
0,90
0,85
0,80
0,75
Batch size 2
Batch size 3
Rule-Based Method
Fuzzy-Q Learning Method


90% of P(High) After Convergence, Batch Arrival Process With Chanaging λ Scenarios
Blocking Probabilities
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
Batch size 2
Batch size 3
Batch size 4
Rule-Based Method
Fuzzy-Q Learning Method


90% of P(Low) After Convergence, Batch Arrival Process With Chanaging λ Scenarios
Blocking Proability
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
Batch size 2
Batch size 3
Batch size 4
Rule-Based Method
Fuaay Q Learning Method

Appendix B

In this section, I have provided the Delphi codes used as the simulator. The codes are generated by in Delphi 5 and only two main programs are presented here. For each specific scenario a unit, was added to simulate required changes.

➢ Rule-Based Method

```
program clouding_ifthen;

uses
  SysUtils,
  Math,
  Procedures-IfThen in '..\Clouding Update3\procedures-IfThen.pas',
  consTypeVar in '..\Clouding Update3\consTypeVar.pas',
  statistics_ifthen in '..\Clouding with statistics\statistics_ifthen.pas',

var
  results_1, results_2                                              :
textfile;

{---------------------------------------------------------------------}
begin
  RandSeed := 123456789;

  Assignfile(results_1,'h:\results_1 (' + ParamStr(1) + ' ' + ParamStr(2) + '
' + ParamStr(3) + ' ' + ParamStr(4) + ' ' + ParamStr(5) + ' ' + ParamStr(6) +
' ' + ParamStr(7) + ' ).txt');
  Rewrite(results_1);
    Assignfile(results_2,'h:\results_2 (' + ParamStr(1) + ' ' + ParamStr(2) +
' ' + ParamStr(3) + ' ' + ParamStr(4) + ' ' + ParamStr(5) + ' ' + ParamStr(6)
+ ' ' + ParamStr(7) + ' ).txt');
  Rewrite(results_2);

  Initialisations;
{Setting Parameters}
  lambda                 :=  StrToFloat(ParamStr(1));
  AvgJobDuration         :=  StrToFloat(ParamStr(2));
  percentHigh            :=  StrToFloat(ParamStr(3));
  capacity.Total         :=  StrToInt  (ParamStr(4));
  capacity.Share         :=  StrToInt  (ParamStr(5));
  Beta                   :=  StrToFloat(ParamStr(6));
  Observationtime        :=  StrToFloat  (ParamStr(7));
  statisticinterval      :=  StrToFloat  (ParamStr(8));
  PLowtarget             :=  StrToFloat(ParamStr(9));
  PHightarget            :=  StrToFloat(ParamStr(10));


  tijd                     := 0.0;
  tijdNextArrival          := tijd + SampleExponential(lambda);
  tijdNextDeparture        := MaxExtended;
  tijdNextOptimisation     := Observationtime;
  tijdNextstatistic        := statisticinterval;
  tijdchangeincapacity    := Observationtime * 10000;
repeat
     if (tijdNextArrival < tijdNextDeparture) and (tijdNextArrival <
     tijdNextOptimisation) and (tijdNextArrival < tijdNextstatistic) then
     {Next Occurrence is Arrival}
      begin
        tijd            := tijdNextArrival;
        tijdNextArrival := tijd + SampleExponential(lambda);
        CallArrival;
      end
      else if (tijdNextDeparture < tijdNextArrival) and (tijdNextDeparture <
      tijdNextOptimisation))and (tijdNextDeparture < tijdNextstatistic)then
           {Next Occurrence is Departure}

            begin
              tijd            := tijdNextDeparture;
              CallDeparture;
            end
      else if (tijdNextOptimisation < tijdNextArrival) and (tijdNextOptimisation <
      tijdNextDeparture) and (tijdNextOptimisation < tijdNextstatistic) then
           {Next Occurrence is Optimization (Happens every 100seconds)}
            begin
                   tijd             :=  tijdNextOptimisation;
                   PLow  := blocks.Low / arrivals.Low;
                   PHigh := blocks.High / arrivals.High;
                   reward := -PLow - (Beta * PHigh);
                   blocks.cumLow       :=   blocks.cumLow + blocks.Low;
                  blocks.cumHigh      :=   blocks.cumHigh + blocks.High;
                  arrivals.cumLow     :=   arrivals.cumLow + arrivals.Low;
                  arrivals.cumHigh    :=   arrivals.cumHigh + arrivals.High;
                   blocks.changeLow    :=   blocks.changeLow + blocks.Low;
                  blocks.changeHigh   :=   blocks.changeHigh + blocks.High;
                  arrivals.changeLow  :=   arrivals.changeLow + arrivals.Low;
                   arrivals.changeHigh := arrivals.changeHigh + arrivals.High;
                  PLowcum             := blocks.cumLow / arrivals.cumLow;
                  PHighcum            := blocks.cumHigh / arrivals.cumHigh;
                  rewardcum           := -PLowcum - (Beta * PHighcum);

                  inc (index);
                  rewardcumMatrix[index]:= -rewardcum;

      writeln(results_1,tijd:20:10,capacity.Total :20:10,
      capacity.Share              :20:10,-reward :20:10, -rewardcum :20:10, -
      rewardchange :20:10 );

              callOptimisation;
              tijdNextOptimisation := tijd + Observationtime;

    end
else        {Next Occurrence is Deriving Statistics (Happens every 300seconds)}

            begin
                   tijd := tijdNextstatistic;
                   PLowsta   := (blocks.staLow / arrivals.staLow);
                   PHighsta  := (blocks.staHigh / arrivals.staHigh);
                   Rewardsta := PLowsta + (Beta * PHighsta);
                    if (PLowsta < PLowtarget ) then  indicatorLow  := indicatorLow
             +1;
                   if (PHighsta <PHightarget )then indicatorHigh := indicatorHigh
      +1;
        if (Rewardsta < PLowtarget + (Beta * PHightarget) )  then
indicatorReward :=              indicatorReward + 1;
        if (PLowsta  < PLowtarget ) and (PHighsta < PHightarget )  and
      (Rewardsta < PLowtarget + (Beta * PHightarget) ) then
            indicatorGeneral := indicatorGeneral +1;
            timefraction :=  indicatorGeneral * statisticinterval;

             writeln(results_2,tijd:20:10, PLowsta:20:10, PHighsta:20:10 ,
      indicatorGeneral:20:10 , (timefraction/tijd):20:10 );
             tijdNextstatistic := tijd + statisticinterval;
             blocks.staLow      := 0;
             arrivals.staLow    := 0;
             blocks.staHigh     := 0;
             arrivals.staHigh   := 0;

            end
      end;
    until (arrivals.Low > 0) and (arrivals.High > 0) and (tijd > (30000 *
Observationtime));

  Closefile(results_1);
  Closefile(results_2);
  {Finding Convergence Time and Converged Cumulative Blocking Value}
  findingfinalreward;

end.

{------------------------------------------------------------------------}
unit procedures-IfThen;
interface
uses
 Math,
 consTypeVar;

function SampleExponential(rate :extended) :extended;
procedure Initialisations;
procedure CallArrival;
procedure CallDeparture;
procedure CallOptimisation;

implementation

{---------------------------------------------------------------------}
{ Poisson Distribution Genetrator}
function SampleExponential(rate :extended) :extended;

begin
  SampleExponential := (-1 / rate) * ln(Random);
end;

{--------------------------------------------------------------}

procedure Initialisations;
{Initialize Values and Matrixes to Zero and Assigning Actions' Values}
var
  i :longword;

begin
    present.High := 0;    present.Low := 0;    present.HighandLow  := 0;
    arrivals.High := 0;   arrivals.Low := 0;   arrivals.HighandLow  := 0;
    accepts.High := 0;    accepts.Low := 0;    accepts.HighandLow  := 0;
    blocks.High := 0;     blocks.Low := 0;     blocks.HighandLow  := 0;
    departures.High := 0; departures.Low := 0;  departures.HighandLow := 0;
    Action1 := 1;        Action2 := 0;                Action3 :=-1;

    for i := 1 to MaxNumCalls do
   begin
       calls[i].departure_tijd := MaxExtended;
       calls[i].callclass      := None;
   end;

end;

{---------------------------------------------------------------------}

procedure CallArrival;
var
  i :longword;

begin
  if (Random < percentHigh) then                   {Arrived Call is
Highclass}
    begin
      if (present.HighandLow < capacity.Total) then {Call is Accepted}
      begin
```

```
           arrivals.High := arrivals.High +1;
           accepts.High := accepts .High +1;
           present.High := present .High +1 ;
           present.HighandLow := present .HighandLow +1 ;


calls[present.HighandLow].departure_tijd := tijd + SampleExponential(1 /
AvgJobDuration);

calls[present.HighandLow].callclass      := High;
      end
      else
      begin                                       {Call is Blocked}
             arrivals.High := arrivals.high +1;
             blocks.High:= blocks.High +1;
      end;
    end
  else                                          {Arrived Call is
Lowclass}
    begin
      if (present.HighandLow< capacity.Share) then   {Call is Accepted}
      begin
             arrivals.Low := arrivals.Low+1;
             accepts .Low := accepts .Low +1;
             present .Low := present .Low +1;
             present.HighandLow := present.HighandLow+1;
calls[present.HighandLow].departure_tijd:=
tijdSampleExponential(1/AvgJobDuration);

calls[present.HighandLow].callclass:= Low;
    end

      else
             begin                                {Call is Blocked}
                 arrivals.Low := arrivals.Low  +1;
                  blocks  .Low := blocks  .Low +1;
              end;
    end;

  {Set New timeNextDeparture}
tijdNextDeparture := MaxExtended; for i := 1 to present.HighandLow do
tijdNextDeparture := min(tijdNextDeparture,calls[i].departure_tijd);
end;

{----------------------------------------------------------------}

procedure CallDeparture;
var
  i,j :longword;

begin
 {Find Call}
  i := 0; repeat inc(i) until (Abs(calls[i].departure_tijd - tijd) <
0.000001);
 {Administration}
  if (calls[i].callclass  = High)            then
  begin
       departures.High :=  departures.High +1;
      present.High:= present.High -1;
      present.HighandLow := present.HighandLow -1;
   end


  else
       begin
             departures.Low:=departures.Low+1;
             present.Low:=present.Low-1;
             present.HighandLow:= present.HighandLow-1;
          end;

  {Remove Departed Call Record}
  for j := i to present.HighandLow do calls[j] := calls[j + 1];
  calls[present.HighandLow + 1].departure_tijd := MaxExtended;
  calls[present.HighandLow + 1].callclass       := None;

  {Set New timeNextDeparture}
  tijdNextDeparture := MaxExtended; for i := 1 to present.HighandLow do
tijdNextDeparture := min(tijdNextDeparture,calls[i].departure_tijd);
end;

{------------------------------------------------------------------}

procedure CallOptimisation;

var
  i          :longword;
  results    : textfile;

begin

       if ( Beta - 0.5 < PLow / (PHigh+ 0.00000000000001)) and (PLow /
       (PHigh+ 0.00000000000001) < Beta + 0.5)   then begin
       {No Action}
      blocks.Low    :=0;
      arrivals.Low  :=0;
      blocks.High   :=0;
       arrivals.High :=0;
      capacity.Share :=  Max(0,Min(capacity.Total,capacity.Share
+Action2));
end
   else
   if (PLow / (PHigh+ 0.00000000000001) < Beta - 0.5)      then
begin
{Decrease C.Shared}
      blocks.Low     :=0;
       arrivals.Low   :=0;
       blocks.High    :=0;
       arrivals.High  :=0;
      capacity.Share :=  Max (0 , Min (capacity.Total,capacity.Share +
Action3)); end
    else
   if (PLow / (PHigh+ 0.00000000000001) > Beta + 0.5)         then
begin
{Increase C.Shared}
       blocks.Low   :=0;
       arrivals.Low  :=0;
      blocks.High   :=0;
       arrivals.High :=0;
      capacity.Share :=  Max (0 , Min (capacity.Total,capacity.Share +
Action1)); end;

end;

end.
```

➢ Fuzzy-Q Learning Method

```
program clouding_RLchange;

uses
  SysUtils,
  Math,
  procedures_RL in '..\Clouding Update3\procedures_RL.pas',
  consTypeVar_RL in '..\Clouding Update3\consTypeVar_RL.pas',
  Statistics in 'Statistics.pas';


{---------------------------------------------------------------------------}

begin
  RandSeed := 123456789;

Assignfile(results,'h:\results (' + ParamStr(1) + ' ' + ParamStr(2) + ' ' +
ParamStr(3) + ' ' + ParamStr(4) + ' ' + ParamStr(5) + ' ' + ParamStr(6) + ' '
+ ParamStr(7) + ' ' + ParamStr(8) + ' ' + ParamStr(9) + ' ' + ParamStr(10) +
'' + ParamStr(11) + ').txt');
  Rewrite(results);
  Initialisations;
  Initialise_RL;
{Setting Parameters}
  lambda                    :=  StrToFloat(ParamStr(1));
  AvgJobDuration            :=  StrToFloat(ParamStr(2));
  percentHigh               :=  StrToFloat(ParamStr(3));
  capacity.Total            :=  StrToInt  (ParamStr(4));
  capacity.Share            :=  StrToInt  (ParamStr(5));
  Beta                      :=  StrToFloat(ParamStr(6));
  gama                      :=  StrToFloat(ParamStr(7));
  learningrate              :=  StrToFloat(ParamStr(8));
  experimentrate            :=  StrToFloat(ParamStr(9));
  observationwindow         :=  StrToInt(ParamStr(10));
  statisticinterval         :=  StrToFloat(ParamStr(11));
  PLowtarget                :=  StrToFloat(ParamStr(12));
  PHightarget               :=  StrToFloat(ParamStr(13));

  tijd                  := 0.0;
  tijdNextArrival       := tijd + SampleExponential(lambda);
  tijdNextDeparture     := MaxExtended;

  tijdNextOptimisation := observationwindow;
  tijdNextstatistic    := statisticinterval;
  Changeincapacitytime :=observationwindow * 10000 ;


  repeat


if (tijdNextArrival < tijdNextDeparture) and (tijdNextArrival <
tijdNextOptimisation) and (tijdNextArrival < tijdNextstatistic) then
      begin             {Next Occurrence is Arrival}
        tijd            := tijdNextArrival;
        tijdNextArrival := tijd + SampleExponential(lambda);
        Arrival;
      end
     else if (tijdNextDeparture < tijdNextArrival) and (tijdNextDeparture <
tijdNextOptimisation) and (tijdNextDeparture < tijdNextstatistic) then
      begin            {Next Occurrence is Departure}
        tijd            := tijdNextDeparture;
        Departure;
      end
 else if (tijdNextOptimisation < tijdNextDeparture) and (tijdNextOptimisation
< tijdNextArrival) and (tijdNextOptimisation < tijdNextstatistic) then
      begin        {Next Occurrence is Optimization Happening every
100seconds}
             tijd := tijdNextOptimisation;
             Optimisation;
             tijdNextOptimisation := tijd + observationwindow;
             arrivals.High := 0;   arrivals.Low := 0;
              accepts.High := 0;    accepts.Low := 0;
              blocks.High := 0;     blocks.Low := 0;
    end


    else
{Next Occurrence is Deriving Statistics Happening every 300seconds}
    begin

           tijd := tijdNextstatistic;
           PLowsta   := (blocks.staLow / arrivals.staLow);
           PHighsta  := (blocks.staHigh / arrivals.staHigh);
          Rewardsta := PLowsta + (Beta * PHighsta);
          if (PLowsta  < PLowtarget )then  indicatorLow  := indicatorLow +1;
         if (PHighsta < PHightarget ) then indicatorHigh := indicatorHigh
+1;
      if (PLowsta  < PLowtarget ) and (PHighsta < PHightarget ) then
      indicatorGeneral := indicatorGeneral +1;
      timefraction:= statisticinterval * indicatorGeneral;
      tijdNextstatistic := tijd + statisticinterval;
      blocks.staLow      := 0;
      arrivals.staLow    := 0;
      blocks.staHigh     := 0;
      arrivals.staHigh   := 0;

      end;
    until (arrivals.Low > 0) and (arrivals.High > 0) and (tijd > (30000 *
observationwindow));

  Closefile(results);

  findingfinalreward;

end.
{-----------------------------------------------------------------------}
       unit procedures_RL;

       interface


       uses
        Math,
        consTypeVar_RL,
        ReinforcementLearning;
        procedure Initialisations;
        function SampleExponential(rate :extended) :extended;
        procedure Arrival;
        procedure Departure;




       implementation

       {----------------------------------------------------------------}
```

```
{Poisson Arrival Generator}

function SampleExponential(rate :extended) :extended;

begin
  SampleExponential := (-1 / rate) * ln(Random);
end;


{-----------------------------------------------------------}
procedure Initialisations; {Initializing Values and Matrixes to Zero}


var
  i :longword;

begin

 present.High    := 0;    present.Low   := 0;    present.HighandLow   :=
0;
arrivals.High    := 0;   arrivals.Low   := 0;   arrivals.HighandLow   :=
0;
accepts.High     := 0;    accepts.Low   := 0;    accepts.HighandLow   :=
0;
blocks.High      := 0;     blocks.Low   := 0;     blocks.HighandLow   :=
0;
departures.High := 0; departures.Low   := 0; departures.HighandLow   :=
0;
blocks.cumLow    := 0;  blocks.cumHigh := 0;     arrivals.cumHigh    :=
0;
arrivals.cumHigh:= 0;
  for i := 1 to MaxNumCalls do
       begin
              calls[i].departure_tijd := MaxExtended;
             calls[i].callclass      := None;
       end;

end;

{----------------------------------------------------------------}

procedure Arrival;

var
  i :longword;

begin
  if (Random < percentHigh) then         {Arrived Call is Highclass}
      begin

      if (present.HighandLow < capacity.Total) then {Call is Accepted}
      begin
              arrivals.High := arrivals.High +1;
               accepts.High  := accepts .High +1;
               present.High  := present .High +1 ;
               present.HighandLow := present .HighandLow +1 ;

              calls[present.HighandLow].departure_tijd := tijd +
              SampleExponential(1 / AvgCallDuration);
```

```
                calls[present.HighandLow].callclass      := High;
         end
         else                                          {Call is
   Blocked}
         begin
                arrivals.High := arrivals.High +1;
               blocks.High:= blocks.High +1;
         end;
       end
     else                                     {Arrived Call is
   Lowclass}
       begin

   if (present.HighandLow< capacity.Share) then   {Call is Accepted}
   begin
          arrivals.Low := arrivals.Low+1;
          accepts .Low := accepts .Low +1;
          present .Low := present .Low +1;
          present.HighandLow := present.HighandLow+1;
          calls[present.HighandLow].departure_tijd      := tijd +
          SampleExponential(1 / AvgCallDuration);

          calls[present.HighandLow].callclass           := Low;
   end
         else     {Call is Blocked}
   begin
          arrivals.Low := arrivals.Low  +1;
          blocks  .Low := blocks  .Low +1;
   end;

end;

     {set new timeNextDeparture}

   tijdNextDeparture := MaxExtended; for i := 1 to present.HighandLow do
   tijdNextDeparture := min(tijdNextDeparture,calls[i].departure_tijd);
   end;

   {--------------------------------------------------------}

   procedure Departure;

   var
     i,j :longword;

   begin
     {Find Call}
     i := 0; repeat inc(i) until (Abs(calls[i].departure_tijd - tijd) <
   0.000001);

     {Administration}
     if (calls[i].callclass  = High)  then
     begin
          departures.High :=       departures.High +1;
          present.High:= present.High -1;
          present.HighandLow := present.HighandLow -1;
     end
```

```
  else
begin
       departures.Low:=departures.Low+1;
       present.Low:=present.Low-1;
       present.HighandLow:= present.HighandLow-1;
 end;

  {Remove Departed Call Record}

  for j := i to present.HighandLow do

       calls[j]   := calls[j + 1];
      calls[present.HighandLow + 1].departure_tijd := MaxExtended;
      calls[present.HighandLow + 1].callclass        := None;

  {Set New timeNextDeparture}
  tijdNextDeparture := MaxExtended; for i := 1 to present.HighandLow
do tijdNextDeparture :=
min(tijdNextDeparture,calls[i].departure_tijd);
end;



end.

{--------------------------------------------------------------------}
unit ReinforcementLearning;

interface

uses
  SysUtils,
  Math,
  consTypeVar_RL;

function DetermineMembershipArray (P :extended) :tMembershipArray;
function DetermineMembershipMatrix(mu1,mu2 :tMembershipArray)
:tMembershipMatrix;
function DetermineSetOfActions (mu: tMembershipMatrix) :
tSetOfActions;
Function CalculatingAction (SAC: tSetOfActions) :Integer;
Procedure Qupdating;
Procedure Qualitycalculating;
procedure Optimisation;
procedure Initialise_RL;


implementation

{-------------------------------------------------------------------}

procedure Initialise_RL;

var
  i,j,c :integer;

begin
  {Determine Fuzzification Sub-State Borders}
```

```
     for i := 1 to NumberOfBorders do border[i] := (i - 1) /
(NumberOfBorders - 1);

     for i := 1 to NumberOfBorders do
       begin
             mu.High  [i] :=0;
              mu.Low   [i]  :=0;
        end;

      for i := 1 to NumberOfBorders do
      for j := 1 to NumberOfBorders do

      mu.HighLow [i,j] := 0;


      for i := 1 to NumberOfBorders do
      for j := 1 to NumberOfBorders do
      for c := 1 to 7               do
      begin
      Q[i,j,c]   :=0;
      Qold[i,j,c]:=0;
      end;

{Define Atomic Actions}
      AtomicActions[0] := -3;
      AtomicActions[1] := -2;
      AtomicActions[2] := -1;
      AtomicActions[3] :=  0;
      AtomicActions[4] :=  1;
      AtomicActions[5] :=  2;
      AtomicActions[6] :=  3;
end;

{-----------------------------------------------------------------------
-}

procedure Optimisation;

var
a,i,j :integer;
action :extended;

begin
       PLow  := blocks.Low / arrivals.Low;
       PHigh := blocks.High / arrivals.High;
       reward:= - PLow - (Beta * PHigh);
       blocks.cumLow    :=   blocks.cumLow + blocks.Low;
       blocks.cumHigh   :=   blocks.cumHigh + blocks.High;
       arrivals.cumLow  :=   arrivals.cumLow + arrivals.Low;
       arrivals.cumHigh :=   arrivals.cumHigh + arrivals.High;
       PLowcum          := blocks.cumLow / arrivals.cumLow;
       PHighcum         := blocks.cumHigh / arrivals.cumHigh;
       rewardcum        := - PLowcum - (Beta * PHighcum);

       writeln(results,tijd:20:10,  capacity.Share :20:10, -
       reward:20:10,(blocks.Low / arrivals.Low):20:10,(blocks.High /
       arrivals.High):20:10 {, -rewardcum:20:10 });
```

```
{Calculating Membership Degrees & Membership Matrix}
       mu.Low          := DetermineMembershipArray (PLow);
       mu.High         := DetermineMembershipArray (PHigh);
       mu.HighLow      := DetermineMembershipMatrix(mu.High,mu.Low);
{Finding Best Action for Each Sub-State}
       SetOfActions    := DetermineSetOfActions(mu.HighLow);
{Calculating the Action}
       action          := CalculatingAction (SetOfActions);
       capacity.Share  := Min ( capacity.Total , Max (0,
       capacity.Share + action));


       Qualitycalculating;
       Qupdating;


{---------------------------------------------------------------------
}

function DetermineMembershipArray (P :extended) :tMembershipArray;

var
 i    :integer;
 temp :tMembershipArray;

begin
       for i := 1 to NumberOfBorders do temp[i] := 0.0;

       i := 0;
       repeat inc(i) until (P >= border[i]) and (P <= border[i+1]);

       temp[i]   := (border[i+1] - P) / (border[i+1] - border[i]);
       temp[i+1] := (P - border[i])   / (border[i+1] - border[i]);

       DetermineMembershipArray := temp;

end;

{---------------------------------------------------------------------
--}

function DetermineMembershipMatrix(mu1,mu2 :tMembershipArray)
:tMembershipMatrix;

var
  i,j  :integer;
  temp :tMembershipMatrix;

begin
for i:=1 to NumberOfBorders do
  for j:=1 to NumberOfBorders do
    temp[i,j] := mu1[i] * mu2[j];

DetermineMembershipMatrix := temp;

end;

{---------------------------------------------------------------------
}
```

```
function DetermineSetOfActions (mu: tMembershipMatrix) :
tSetOfActions;

var
  i,j,k,m :integer;
  temp    :tSetOfActions;

begin
 k := 0;

 for i:=1 to NumberOfBorders do
   for j:=1 to NumberOfBorders do
     if (mu [i,j] > EPSILON) then
       begin
         inc(k);

         if random < experimentrate then
           begin
             temp[k].actionid      := random(6);
             temp[k].mu            := mu[i,j];
             temp[k].quality       := Q[i,j,temp[k].actionid];
           end
         else
           begin
             qmax := -999999.99;
             for m:= 0 to 6 do
               begin
                 if Q[i,j,m] > qmax then
                   qmax := Q[i,j,m];
               end;

             for m:= 0 to 6 do ArrayOfBest[m] := 99999;
             NumOfBest := 0;
             for m:= 0 to 6 do
               if (Abs(Q[i,j,m] - qmax) < 0.000001) then
                 begin
                   NumOfBest := NumOfBest + 1;
                   ArrayOfBest[NumOfBest - 1] := m;
                 end;

             ChosenBest := Random(NumOfBest);
             temp[k].actionid       := ArrayOfBest[ChosenBest];
             temp[k].mu             := mu[i,j];
             temp[k].quality        := Q[i,j,temp[k].actionid];
           end;
       end;

  DetermineSetOfActions := temp;
end;

{--------------------------------------------------------------------}



Function CalculatingAction (SAC: tSetOfActions) :Integer;
var
  k :integer;
  l :integer;
  action :extended;
  begin
  action := 0;
   for k:=1 to 4 do
    begin
      l      := SetOfActions[k].actionid;
      action := action + SetOfActions[k].mu* AtomicActions[l];
    end;

  CalculatingAction := Round(action);
end;

{-------------------------------------------------------------------}

Procedure Qualitycalculating;
var m,i,j,k,l,mmax: integer;


begin
k := 0;
for l := 1 to 4 do
  begin
      Maxquality[l].actionid       := 0;
      Maxquality[l].quality        := 0;
      Maxquality[l].mu             := 0;
  end;

for i:=1 to NumberOfBorders do
  for j:=1 to NumberOfBorders do
       if mu.HighLow[i,j] > EPSILON then
 begin

    qmax := -999999.99;
             for m:= 0 to 6 do
               if Q[i,j,m] > qmax then
                 begin
                   qmax := Q[i,j,m];
                   mmax := m;
                 end;





                   inc(k);
                   Maxquality[k].actionid       := mmax;
                   Maxquality[k].quality        := Q[i,j,mmax];
                   Maxquality[k].mu             := mu.HighLow[i,j];

                   end;

               actionquality.Old := 0;
               actionquality.New := 0;
              for l:= 1 to 4 do
              begin

                   actionquality.New  := actionquality.New +
(Maxquality[l].quality * Maxquality[l].mu);
                   actionquality.Old := actionquality.Old +
 (SetOfActionsOld[l].quality* SetOfActionsOld[l].mu);
              end;
              deltaq             := reward + (gama*actionquality.New)
- actionquality.Old;



 end;
{------------------------------------------------------}

 Procedure Qupdating;
 var x,y,c,f : integer;
 begin

c:= 0;
for x:=1 to NumberOfBorders do
  for y:=1 to NumberOfBorders do
    if (mu.HighLowOld[x,y] > EPSILON) then
      begin
      inc(c);
      f := SetOfActionsOld[c].actionid;
      Q[x,y,f] :=  Q[x,y,f] + (learningrate * deltaq *
mu.HighLowOld[x,y]);
      end;



    mu.HighLowOld  := mu.HighLow;
    SetOfActionsOld := SetOfActions;

end;

{-----------------------------------------------------------------}

end.
```