

OPTIMIZING SERVICE RATE OF A SINGLE SERVER QUEUE USING
REINFORCEMENT LEARNING

by

Furkan Can Ercan

B.S., Industrial Engineering, Marmara University 2021

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering
Boğaziçi University

2024

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Aybek Korugan, in this long and hard journey he was always positive and supportive. I am grateful for his guidance and belief in me during my research. I would like to thank my family my mother Güler Ercan, my father Faruk Ercan and my brother Berat Ercan who were always there for me during this period of hard work. I would also like to thank my friends and colleagues who believed in me more than I did. As a final note I would like to thank Mustafa Ertuğrul, who was my primary school teacher and the reason why everything started.

ABSTRACT

OPTIMIZING SERVICE RATE OF A SINGLE SERVER QUEUE USING REINFORCEMENT LEARNING

This thesis investigates the optimization of queueing control problems using reinforcement learning (RL), more specifically Q-learning. The study focuses on solving a service rate balancing problem on M/M/1/K queueing systems and then extends to M/G/1/K problems. By using a simulation environment, the Q-Learning Algorithm is trained and compared with optimal results derived from Markov Decision Process (MDP) models. The results indicate that Q-Learning can effectively solve the M/M/1/K problems by creating results that are very close to the optimum. The study explores the effectiveness of different exploration strategies, such as ϵ -Greedy, Boltzmann, and Upper Confidence Bound (UCB) exploration, to improve performance and the convergence rate of Q-learning algorithms. Key findings indicate that Boltzmann exploration strategy perform better than the other strategies, providing a good balance between exploration and exploitation. The study also highlights the sensitivity of Q-learning Algorithm to hyperparameter values, emphasizing the need for careful tuning to achieve optimal performance. The extension of Q-learning to M/G/1/K queueing systems demonstrates the algorithm's adaptability, and the limits and sensitivities of this adaptability. The extension also depicts that embedding model knowledge to the Q-Learning algorithm may not create better results. In conclusion, this thesis validates the potential of reinforcement learning, particularly Q-learning, as an effective tool for optimizing queueing control problems, offering a flexible and adaptable alternative to traditional methods. The results encourage additional research and development of reinforcement learning methods to optimize complex systems and solve harder queueing control problems.

ÖZET

PEKİŞTİRMELİ ÖĞRENMEYİ KULLANARAK TEK BİR SUNUCU KUYRUĞUNUN HİZMET HIZINI OPTİMİZE ETME

Bu tez, kuyruk kontrol problemlerinin optimizasyonunu pekiştirmeli öğrenme (RL) ve daha spesifik olarak Q-öğrenme yöntemini kullanarak araştırmaktadır. Çalışma, M/M/1/K kuyruk sistemlerinde hizmet hızı dengeleme problemini çözmeye odaklanmakta ve daha sonra M/G/1/K problemine genişletilmektedir. Simülasyon ortamı kullanılarak, Q-Öğrenme Algoritması eğitilmiş ve Markov Karar Süreci (MDP) modellerinden elde edilen optimal sonuçlarla karşılaştırılmıştır. Sonuçlar, Q-Öğrenme'nin M/M/1/K problemlerini optimuma çok yakın sonuçlar üreterek etkili bir şekilde çözebileceğini göstermektedir. Çalışma, Q-öğrenme algoritmalarının performansını ve yakınsama hızını artırmak için farklı keşif stratejilerinin, etkinliğini incelemektedir. Bulgular, Boltzman keşif stratejisinin diğer stratejilere göre daha iyi performans gösterdiğini ve keşif ile sömürü arasında iyi bir denge sağladığını göstermektedir. Çalışma ayrıca, Q-öğrenme Algoritması'nın hiperparametre değerlerine karşı duyarlılığını vurgulayarak, optimum performansa ulaşmak için dikkatli bir seçim yapılması gerektiğini belirtmektedir. Q-öğrenmenin M/G/1/K kuyruk sistemlerine genişletilmesi, algoritmanın adapte olma yeteneğini ve bu yeteneğin sınırlarını ile duyarlı olduğu noktaları göstermektedir. Genişletme ayrıca, Q-Öğrenme algoritmasına model bilgisinin dahil edilmesinin daha iyi sonuçlar yaratmayabileceğini göstermektedir. Sonuç olarak, bu tez, özellikle Q-öğrenmenin, kuyruk kontrol problemlerinin optimizasyonu için etkili bir araç olarak potansiyelini doğrulamakta ve geleneksel yöntemlere esnek ve uyum sağlama gücü yüksek bir alternatif sunmaktadır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZET	v
LIST OF FIGURES	viii
LIST OF TABLES	ix
LIST OF SYMBOLS	xiv
LIST OF ACRONYMS/ABBREVIATIONS	xvi
1. INTRODUCTION	1
2. LITERATURE REVIEW AND BACKGROUND	7
2.1. Studies of Queueing Network Models	7
2.2. Studies on Queueing System Models	11
2.3. Background on Q-Learning	17
3. METHODOLOGY	23
3.1. Problem Definition and Objective	23
3.2. Model Development	24
3.2.1. MDP Model	24
3.2.2. The Q Learning Algorithm, Convergence and Convergence Cri- terion	26
3.2.3. Convergence of The Q Learning Algorithm and Hyperparameter Search	33
3.2.3.1. Boltzman Exploration Strategy	36
3.2.3.2. The Upper Confidence Bound Strategy	38
3.2.3.3. ϵ -Greedy Strategy	39
3.2.4. Q-Learning Algorithm	39
3.3. Simulation	47
3.3.1. Simulation Verification and Validation	47
4. Numerical Analysis	52
4.1. Parameter Sets and Experimental Design	56

4.2. Results of M/M/1/K Experiments	59
4.2.1. Performance Comparison of the Q-Learning Models with MDP .	59
4.2.2. Performance Evaluations of Q-Learning Models	67
4.2.3. Performance of the Boltzman Strategy	74
4.2.4. Performance of the ϵ -Greedy Strategy	75
4.2.5. Performance of Upper Confidence Bound (UCB) Strategy	76
4.3. Performance Results of M/G/1/K Instances	77
4.4. Overall Findings and Insights	92
5. CONCLUSION	94
REFERENCES	96
APPENDIX A: TABLES REGARDING SIMULATION VERIFICATION . . .	102

LIST OF FIGURES

Figure 2.1.	Reinforcement Learning Domain Classification	19
Figure 3.1.	Probabilities of selecting actions for different τ values	37
Figure 3.2.	Simulation structure diagram	49



LIST OF TABLES

Table 1.1.	Comparison of the advantages and disadvantages of RL methods and traditional approach.	5
Table 2.1.	Summary of the studies and their approaches in the literature. . .	15
Table 3.1.	Performance Measures for $\rho = 0.9$ and 90% confidence.	50
Table 3.2.	M/M/1/K Probability values vs simulation values for $K = 10$ and $\rho = 1$	51
Table 4.1.	Data sizes for different simulation runs.	55
Table 4.2.	Experimental setup of the problems.	56
Table 4.3.	Hyperparameter settings for different strategies.	57
Table 4.4.	Summary of the experimented problems.	59
Table 4.5.	Comparison of the results of the Boltzman Strategy and MDP. . .	60
Table 4.6.	Comparison of of the results of ϵ -Greedy Strategy and MDP. . . .	61
Table 4.7.	Comparison of of the results of UCB Strategy and MDP.	62
Table 4.8.	Comparison of MDP policies and the best policies of the Boltzman Strategy.	64

Table 4.9.	Comparison of MDP policies and the best policies of the ϵ -Greedy Strategy.	65
Table 4.10.	Comparison of MDP policies and the best policies of the UCB Strategy.	66
Table 4.11.	Comparison of exploration strategies for the instances with $\rho = 0.75$	67
Table 4.12.	Comparison of exploration strategies for the instances with $\rho = 0.9$	68
Table 4.13.	Comparison of exploration strategies for the instances with $\rho = 0.95$	69
Table 4.14.	Average time per episode of strategies for the instances with $\rho = 0.75$	71
Table 4.15.	Average Time Per Iteration of Strategies for the instances with $\rho = 0.90$	72
Table 4.16.	Average Time Per Iteration of Strategies for the instances with $\rho = 0.95$	73
Table 4.17.	Boltzmann Strategy best performing models.	75
Table 4.18.	ϵ -Greedy Strategy best performing models.	76
Table 4.19.	UCB Strategy Best Performing Models	77
Table 4.20.	MDP-Informed model's performance in comparison with the MDP policy in Non-Markovian setting with 95% CI for the instances with $= 0.75$	80

Table 4.21.	Ordinary Q-Learning model's performance in comparison with MDP policy in Non-Markovian setting with 95% CI for the instances with $\rho = 0.75$	81
Table 4.22.	MDP-Informed model's performance in comparison with MDP policy in Non-Markovian setting with 95% CI for the instances with $\rho = 0.90$	82
Table 4.23.	Ordinary Q-Learning model's performance in comparison with MDP policy in Non-Markovian setting with 95% CI for the instances with $\rho = 0.90$	83
Table 4.24.	MDP-Informed Model's performance in comparison with MDP policy in Non-Markovian Setting with 95% CI for the instances with $\rho = 0.95$	84
Table 4.25.	Ordinary Q-Learning Model's performance in comparison with MDP policy in Non-Markovian setting with 95% CI for the instances with $\rho = 0.95$	85
Table 4.26.	Comparison of performance for MDP-Informed models and ordinary models with 95% Confidence Intervals and $\rho = 0.75$	86
Table 4.27.	Comparison of cost differences and iterations for MDP-Informed models and ordinary models for the instances with $\rho = 0.75$	87
Table 4.28.	Comparison of performance for MDP-Informed models and ordinary models with 95% Confidence Intervals and $\rho = 0.90$	88
Table 4.29.	Comparison of cost differences and iterations for MDP-Informed models and ordinary models for the instances with $\rho = 0.90$	89

Table 4.30.	Comparison of performance for MDP-Informed models and ordinary models with 95% Confidence Intervals and $\rho = 0.95$	90
Table 4.31.	Comparison of cost differences and iterations for MDP-Informed models and ordinary models for the instances with $\rho = 0.95$	91
Table A.1.	Verification of Parameters M/M/1 Queue with $\rho = 0.25$ and 95% Confidence	102
Table A.2.	Verification of Parameters M/M/1 Queue with $\rho = 0.25$ and 90% Confidence	102
Table A.3.	Verification of Parameters M/M/1 Queue with $\rho = 0.5$ and 95% Confidence	103
Table A.4.	Verification of Parameters M/M/1 Queue with $\rho = 0.5$ and 90% Confidence	103
Table A.5.	Verification of Parameters M/M/1 Queue with $\rho = 0.75$ and 95% Confidence	103
Table A.6.	Verification of Parameters M/M/1 Queue with $\rho = 0.75$ and 90% Confidence	104
Table A.7.	Verification of Parameters M/M/1 Queue with $\rho = 0.8$ and 95% Confidence	104
Table A.8.	Verification of Parameters M/M/1 Queue with $\rho = 0.8$ and 90% Confidence	104

Table A.9.	Verification of Parameters M/M/1 Queue with $\rho = 0.90$ and 95% Confidence	105
Table A.10.	Verification of Parameters M/M/1 Queue with $\rho = 0.90$ and 90% Confidence	105
Table A.11.	M/M/1/K Probability Values vs Simulation Values for $K = 10$ and $\rho = 1$ 95% Confidence	105
Table A.12.	M/M/1/K Probability Values vs Simulation Values for $K = 10$ and $\rho = 1$ 90% Confidence	106
Table A.13.	M/M/1/K Probability Values vs Simulation Values for $K = 15$ and $\rho = 1$ 95% Confidence	106
Table A.14.	M/M/1/K Probability Values vs Simulation Values for $K = 15$ and $\rho = 1$ 90% Confidence	107
Table A.15.	M/M/1/K Probability Values vs Simulation Values for $K = 25$ and $\rho = 1$ 95% Confidence	108
Table A.16.	M/M/1/K Probability Values vs Simulation Values for $K = 25$ and $\rho = 1$ 90% Confidence	109

LIST OF SYMBOLS

$ A(s) $	The cardinality of the set of the possible action in state s
c	The hyperparameter of UCB strategy
c_i	Service cost of the alternative rate i
d_1	The difference between the value function a given problem at step $n + 1$ and n
d_2	The difference between the value function a given problem at step $n + 1$ and n
$\tilde{d}_1$	The difference between the estimated value function a given problem at step $n + 1$ and n
$\tilde{d}_2$	The difference between the estimated value function a given problem at step $n + 1$ and n
h	Holding cost
G	General Distributions which are not Markovian
L	Little's Formula
L_q	Little's Formula for the time spent in queue
M	Markovian Distribution, e.g. Poisson & Exponential Distributions
$M/G/1/K$	A queueing system where inter-arrival times Markovian, service times are general distribution service policy is FCFS and capacity is K
$M/M/1/K$	A queueing system where inter-arrival and service times are Markovian, service policy is FCFS and capacity is K
n	The number of the episode in the Q-Learning Algorithm
$N(s_t, a_t)$	The total number of times action a selected at the state s at time step t
$p(s' s, a)$	Given that the current state s and the action taken is a the probability of going to a state s' in an MDP problem
$Q(s, a)$	The table of estimated state action values in the Q-Learning Algorithm
R	The Reward Set of an MDP Model

r_t	Immediate reward at time t
S	State set of an MDP model
t	The total visits to a given state and action in at time step t an MDP problem
T	Set of Time Steps in an MDP model
$V_n(X)$	Bellman Equations of the MDP model where X is the number of jobs in the system at the time step n
λ	Arrival Rate
μ	Service Rate
Λ	The total rate of Arrival,Service and Staying
π	The policy of an MDP problem
v_π	Value Function under the policy π
$\tilde{v}_\pi$	Estimated Value Function under the policy π
$q(s, a)_\pi$	State Action Values under policy π
$\tilde{q}(s, a)_\pi$	Estiamted state action values under policy π
v_π^*	Value Function under the the optimal policy π^*
$q(s, a)_\pi^*$	State Action Values under the optimal policy π^*
E_π	Expectation under the policy π
α	Discount Rate at an MDP problem
Ω	The set of the all possible sample paths of an MDP problem
ω	The set of a realization of of all possible sample paths of an MDP problem
ϵ	Probability of choosing a non-greedy action under the ϵ - Greedy Strategy
β	Learning rate in the Q-Learning Algorithm
σ	The convergence threshold
ρ	The intensity of a Queue
τ	The temperature hyperparameter of Boltzman Strategy
ω	The average time spent in the system
ω_q	The average time spent in the queue system

LIST OF ACRONYMS/ABBREVIATIONS

FCFS	First come first serve service policy
MDP	Markov Decision Process
RL	Reinforcement Learning
ML	Machine Learning
TD	Temporal Difference
UCB	Upper Confidence Bound

1. INTRODUCTION

Waiting in line is a phenomenon [1] that everybody experiences and dislikes. It is disliked by customers who have to wait to get a service and business owners due to loss of business opportunities. It is due to the fact that after a certain point of discontent, dissatisfaction, and a loss of faith that their turn will come, customers are likely to exit the system and try their luck elsewhere. Not only humans but also materials in production lines, such as car parts, may have to wait to progress to the next stage in the process [2]. Additionally, another very popular place where queueing happens nowadays is the computer networks that provide us social media, online video streaming services, and cloud computing [3, 4]. In computer networks, requests for online video streaming or cloud computing services must wait in queues, ideally for a tolerable amount of time. Queueing systems are fundamental to many more fields other than mentioned above [5], including telecommunications, service industries, and healthcare [6]. Effective queueing control is an important topic of research in operations research since it is essential for maximizing system performance and reducing delays for many of the systems in real life. The fundamental goal of queueing theory is to offer analytical models that forecast and utilize metrics such as wait times, queue lengths, capacity, and alike to implement an effective design for queueing systems that can manage demand unpredictability efficiently [1].

This thesis focuses on the M/M/1/K queue problem, a single-server queue with a finite capacity K. It is a widely studied model in queueing theory due to its practical relevance and tractable analysis. It is often used in manufacturing with buffers, banking queues, and telecommunication systems. [5], [7]. It is also used to analyze more complex models in production and in computer networks [2], [4]. The model represents systems with a limit to the number of customers that can be accommodated, which is a common scenario in many real-world applications, such as call centers, manufacturing lines, and computer networks. Managing such queues with buffers effectively requires balancing the trade-offs between server utilization and customer waiting times, which can be

challenging under dynamic conditions.

The literature on queueing control provides two main approaches to managing M/M/1/K queues: optimal capacity determination and optimal service rate adjustment [1], [8]. The first approach involves determining the optimal capacity K of the system. This method seeks to balance the arrivals of customers with the available service capacity to minimize costs and waiting times. It is especially useful in environments where the arrival rate can be influenced or controlled, such as reservation systems or admission control mechanisms. Effective arrival control can significantly improve system performance by preventing overloading and reducing congestion [1], [9, 10]. The second approach focuses on determining the optimal service rate. This approach aims to adjust the service rate dynamically based on current queue conditions to improve system performance and ensure efficient resource utilization. Service rate control can be implemented in various ways, such as through variable staffing levels, adjustable processing speeds, or flexible scheduling by optimizing the service rate. Organizations can better match their service capacity to fluctuating demand, thereby enhancing overall efficiency and customer satisfaction [1], [11, 12]. Both of these strategies are crucial for effective queue management, as they address different dimensions of the problem and offer complementary solutions to enhance overall system efficiency. The focus of this study is on solving service rate control problems using reinforcement learning (RL) approaches, specifically Q-Learning.

Traditional methods for controlling M/M/1/K queues are the analytical solutions that assume known parameters of arrival and service rates. This is an assumption that may not always be valid in the real-world scenarios. Thus, a practical way to relax this assumption is to combine Markovian Decision Process (MDP) models with parameter estimation techniques such as statistical estimation or supervised learning. Using the estimates of parameters, MDP models can be used to optimize certain parameters of interest under the assumption that Markovian property is valid. Many studies have applied MDP to optimize control mechanisms for various queueing models. [13–17].

Reinforcement learning, a subset of machine learning, has emerged as a powerful tool for solving complex control problems due to its ability to learn optimal policies through interaction with the environment. Since reinforcement learning techniques learn from the consequences of actions rather than labeled input/output pairs as in supervised learning, they are suitable for problems which includes dynamic environments and randomness. Because of the adaptability and flexibility of the RL techniques, they are desirable for handling a variety of operational problems, including those arising from queueing systems. This thesis investigates the possibility of optimizing service rate in M/M/1/K systems through the use of Q-learning, a model-free RL algorithm. Q-learning can adjust to changes in system dynamics without requiring explicit knowledge of the underlying processes, learning directly from interactions within the environment. This capability is especially useful in real-world situations where sudden changes can occur. It can be used to model very complex problems where analytical models can be intractable. Being a model-free algorithm, Q-Learning does not require developing a model of the system, it can be directly used. The algorithm predicts and optimizes the system at the same time with the data it has, which does not need a separate prediction method. It is an online algorithm which can work with the data as it comes in the real time and this makes it very useful in practice. It estimates the state-action values rather than the parameters itself, which can be used to gain insights on different action preferences in different situations. However, Q-learning has drawbacks, such as the problem of the exploration-exploitation trade-off in the learning process and the algorithm's sensitivity to hyperparameter values, which are crucial for the algorithm's performance and require extra effort to be carefully tuned for each problem. The need for the hyperparameters both originate from the model equation and exploration strategies. While model hyperparameters control the algorithm's learning behavior, hyperparameters of exploration strategies control the degree of exploration during the learning phase. There is not one general approach to use to select the best hyperparameter values, as it is problem-dependent. A bad selection of hyperparameter values can lead to long convergence times. Additionally, the data requirements of RL algorithms are much higher than the traditional approach. Yet, there are various methods in literature to solve this problem such as training the model

offline with simulation data and then adapting it to the real environment [18,19], using good exploration strategies [20–22], or any other data augmentation techniques such as state and action perturbations, mix-up and prioritized experience replay which allows us to synthesize new data from the existing data to increase the amount of training data. [18,19], [23]. Another drawback is that the Q-Learning model only depends on the data itself and does not consider causal relations. This situation is both a blessing and a curse. It is a blessing due to the advantages we mentioned above but also a curse as learning rare cases in a system is difficult since the data on rare events are also rare and the model has higher data needs. Table 1.1 summarizes the advantages and disadvantages of the RL methods and the traditional approach.

This thesis addresses the challenge of optimizing queueing control in M/M/1/K systems using reinforcement learning. Specifically, we develop a Q-learning algorithm that can learn optimal control policies without requiring explicit knowledge of the system’s parameters. We focus on the algorithm’s convergence and use three well-known and widely used exploration strategies (ϵ -Greedy exploration, Boltzman exploration and UpperConfidenceBound(UCB)). We explore them with different hyperparameter values for different problem settings with varying utility rates and K values. After finding the best exploration method and the best hyperparameter setting we extend our M/M/1/K model to M/G/1/K model. We model 3 different distributions with varying coefficients of variations. In the extension part, we encode information to the Q-Learning model regarding the environment, by giving the optimal solution obtained with MDP of the M/M/1/K problem as a starting point to the algorithm to use in the M/G/1/K problem. We observe its performance and compare it with the MDP optimal policy of M/M/1/K problem in M/G/1/K setting and the ordinary Q-Learning model in the M/G/1/K setting. In the extension, we do not make any hyperparameter tuning, but we use the best-performing strategy and parameters of the M/M/1/K problem. By doing these we test two things regarding Q-Learning. First, its adaptability to a non-Markovian environment without any hyperparameter tuning is done, and second, the effect of encoding information to a model-free RL algorithm. The data is generated via simulations and the models are trained in the simulations. The obtained

Table 1.1. Comparison of the advantages and disadvantages of RL methods and traditional approach.

Aspect	Reinforcement Learning (RL)	Parameter Estimation & Analytical Formulas
Method	Interaction-based learning (Q-learning)	Estimation of parameters and use of mathematical models
Flexibility	High, adapts to the changes in parameters	Moderate, depends on accuracy of parameter estimates
Data Requirements	High, needs extensive interaction data	Moderate, needs historical or real-time data for estimates
Complexity	Handles complex, high-dimensional spaces	Suitable for simpler, well-modeled systems
Scalability	Scales with neural networks but computationally intensive	Limited by model assumptions and simplifications
Accuracy	High, captures real-time nuances	High for well-understood systems, less for complex ones
Adaptability	Highly adaptable, learns from changes	Less adaptable, needs re-estimation or re-modeling
Robustness	Robust if well-trained, sensitive to hyperparameters	Robust within model assumptions, may be inaccurate otherwise
Convergence	Wrong tuning may create long convergence times	No problem on convergence for stationary processes

policies tested using value iteration for M/M/1/K setting and simulation environments for M/G/1/K setting. The remainder of this thesis is organized as follows.

Chapter 2 provides a literature review of RL methods for queueing problems with classification and a detailed background on the Q-learning algorithm. Chapter 3 describes the detailed definition of the problem, the MDP model and Bellman Equations, together with the development of the Q-learning algorithm, convergence proof, and data collection methods. Chapter 4 presents the numerical analysis of the results of the trials with different exploration strategies, different hyperparameter values and a comparison of the algorithm's results with the optimal MDP policies. At the end of the chapter we share our results on the extension of our method to the M/G/1/K case and summarize our overall findings. Chapter 5 discusses the findings and their implications and summarizes the contributions of this thesis.

2. LITERATURE REVIEW AND BACKGROUND

The literature regarding reinforcement learning in queueing systems covers a wide range of study areas. The studies could be divided into two categories. The first category includes research on complex computer networks, most of which focus on cloud networks and cost-effectively controlling latency on those networks. The complex queue structures found in computer network problems make it challenging to model them using queueing theoretic approaches. Recognizing that, the main goal of these studies is to optimize these queueing networks through model-free or model-based methods of the reinforcement learning domain. The second category includes research on solving well-known problems of queueing theory using the reinforcement learning methods. Despite their elegance and utility, queueing theoretic solutions necessitate precise knowledge of system factors, such as arrival and service time distributions. It is typically not easy to acquire those parameters in real life. The second group of studies focuses on this issue and creates frameworks of reinforcement learning to address such situations. The richness of the reinforcement learning domain is demonstrated by the literature on these two categories. RL techniques can be applied to address various problems, whether your goal is to optimize a very complicated queueing network or a straightforward and well-known queueing theory problem. Due to the purpose of this study, which is to use RL methods to tackle well-known problems of queueing theory, we focus on the studies in the second category.

2.1. Studies of Queueing Network Models

Liu et al. [24] created a model-based reinforcement learning algorithm that learns the optimal policy to minimize average job delay. They evaluate their model's performance in dynamic server allocation and routing problems and claim that their results show an effective performance of their developed algorithm. The challenge of their case is that they work with a countable state Markov Decision Problem in which the state space is unbounded. To overcome that challenge they create an auxiliary bounded

system with bounded state space. They divide the whole state space into 2 parts and while the algorithm learns about the bounded system, a known stabilizing policy is applied to the rest of the states. They claim that when a large enough auxiliary system is created the solution of their framework becomes arbitrarily close to the optimal policy. This is a good example of using a model-based Reinforcement Learning method on complex queueing problems. Their algorithm tries to learn the correct state-transition probability matrix and assumes Markov Property which is the reason that their approach is classified as a model-based method. Their method is based on the fact that the probability of visiting some states in a stable queueing systems is very low. Thus as the auxiliary system gets larger the probability of not learning an important aspect of the problem gets smaller.

Dai et al. [25] focus on multiclass queueing networks with long-run average cost objectives. They claim that these kinds of problems are known as notoriously difficult to solve due to the infinite state space and unbounded costs especially when the objective is to minimize the long-run average cost. They try the dominant state-of-the-art advanced policy gradient methods such as proximal policy optimization to solve these problems. They claim that their extension of the advanced policy gradient methods outperforms state-of-the-art heuristics. They also show that in the cases where the optimal policy can be computed the policies generated by their reinforcement learning algorithm are near optimal. Their paper shows that state-of-the-art Deep Reinforcement Learning methods can be used for problems that are acknowledged as very hard by the literature and can generate very good results. This is a paper that supports the usage of Reinforcement Learning methods for queueing problems.

Bachl et al. [26] develop a reinforcement learning algorithm called Learning Fair Qdisc (LFQ) which is an online learning algorithm that optimizes buffer size for each flow according to a reward function. They train their model offline via data generated by a simulation first and then deploy it to a system. In the live system, the model continues to learn as an online learner. They focus on fair queueing systems and claim that previously to their work, in fair queueing systems, each queue is either

managed by Active Queue Management (AQM) or has a fixed maximum size. The LFQ is an AQM mechanism that learns the system online and manages the sizes of each queue accordingly. They claim that their algorithm assigns optimal queue size to each flow based on its congestion control, delay, and bandwidth. They also claim that their algorithm outperforms other AQM scheduler algorithms. In summary, they create an adaptive AQM scheduler algorithm that learns the system that it operates in, which dynamically learns and assigns optimal buffer sizes to the queues.

Raeis et Al. [27] use the deterministic policy gradient method to ensure quality of service in a tandem queue network by guaranteeing probabilistic upper bounds on the end-to-end delay of the system. The algorithm learns and then controls the service rates in tandem queueing networks to achieve its goal. Their approach is model-free and uses only queue length information of the tandem queues to optimize the network. The interesting part is that they do not use a standard reward criterion such as achieved overall reward to evaluate their algorithm which they claim is hard to interpret and even misleading. Instead, they try to provide certain probability bounds on the end-to-end delay of the system. They claim their controller algorithm ensures explicit probabilistic guarantees for the end-to-end delay of the system. Thus their controller gives a probabilistic guarantee to the customers on how much delay they will encounter which they claim is very important for Quality of Service (QoS) constraints.

Xu et al. [28] develop a reinforcement learning approach to control complex communication networks. They focus on traffic engineering of the networks which aims to minimize the end-to-end delay of a queueing network. They claim that the assumptions of queueing theory is not practical for their case hence they develop a model-free approach. They try deep deterministic policy gradient (DDPG) algorithm however it does not work well for their case. They then develop their Deep Reinforcement Learning Traffic Engineering (DRL-TE) algorithm. They use the DDPG algorithm as a beginning and modify it to suit their problem better. Though they use a model-free approach, their modification of the model involves knowledge related to their problem: Their empirical assessments underscore the superior performance of DRL-TE compared

to traditional benchmarks, particularly in mitigating end-to-end delay and optimizing network utility.

Kim et al. [29] use Deep Q- Network methodology and propose a new Active Queue Management (AQM) scheme based on deep reinforcement learning. They introduce a scaling factor in the reward function to balance the trade-off of queueing delay and throughput. They compare their method with various AQM schemes such as, P-FIFO, FQ-CoDel. They claim that their method provided low delay in most cases while ensuring above average throughput. They claim that their proposed Reinforcement Learning based AQM scheme outperforms other AQM algorithms.

Hsu et al. [30] integrates learning and control with an online learning algorithm in their paper. They mention that the online platforms that make everyone's life easier all depend on efficiently assigning client's tasks to the servers which in turn necessitates the control of emerging queueing system. The problem requires 2 things, learning unknown parameters of arrival, service and pay-off distributions and optimizing the control of the system. They claim that the previous approach in the literature to divide this problem into 2 parts and dealing with them separately creates sub-optimal results. They propose an online learning algorithm that learns the system dynamics and optimize the system at the same time and they show that not only their approach is better than the sub-optimal approaches of the literature but also it gives results that which the difference to the optimal solution is bounded.

Sentenac et al. [31] consider competing queues in a multi-queue and multi-server system. The servers work on a first come first serve methodology and picks uniformly among the packets that arrive at the same time. They evaluate the system in a game theoretic approach, and they show that by referring to the literature, to have a stable system in decentralized systems, one needs to have a ratio of 2 between the arrival and service rates. They then propose a learning algorithm to ensure a stable system when the ratio is larger than one and they claim to do that by ensuring some cooperation between the queues instead of selfishly minimizing their regret individually. They prove

that their algorithm (ADEQUA) ensures the stability when the ratio is larger than one.

2.2. Studies on Queueing System Models

Carner et al. [32] use 2 different approaches to estimate the delays of a queueing network. The model-based approach is a queueing theory inspired regression model while the model free approach is using artificial neural networks. They claim that neural network model performs much better than the queueing theory inspired model. However, the black-box nature of the neural network model does not provide any insights on the queueing model while theoretically inspired regressor provides insights about the underlying queueing structure. This paper shows the well-known advantages and disadvantages of model-free machine learning based models in the queueing domain. Despite performing very well machine learning models do not provide much insight.

Chen et al. [33] develop an online learning algorithm to solve the problem of pricing and sizing in M/G/1 queueing systems. In this paper they compare a Predict Then Optimize (PTO) approach with the online learning algorithm they develop. They claim that predicting the model parameters and using queueing theoretic formulas to optimize is highly sensitive to the prediction of parameters. The proposed online algorithm is called as LiQUAR and works on cycles, it uses the data collected on previous cycles to dynamically adjust the best service fee and rate for the current cycle. This is an online model-free reinforcement learning algorithm that can work in real time. They test their algorithm's convergence and effectiveness both by regret analysis and simulation experiments and they claim that with any initial pricing and servicing policy (μ_0, p_0) the algorithm quickly learns the optimal policy without losing much profit on the learning phase.

Again Chen et al. [34] investigate a queueing system where the controller selects a service fee and service capacity to maximize the cumulative expected profit. They work on G/G/1 setting and express that this problem has no analytical solution as the research is focused on heavy-traffic analysis. In their study they propose an online learning framework which is a model-free gradient based reinforcement learning algorithm. Their algorithm is called GOLiQ and again similar to above paper it works on cycles which dynamically provides policies by using the data generated on previous cycles. The data the algorithm uses consists of the number of customer arrivals, waiting times, and the server's busy times. They claim that the online nature of the algorithm allows the controller to adapt to the situation at hand and make better decisions. They claim that their algorithm is effective by showing the convergence of the algorithm via regret analysis as well as via simulation experiments of different G/G/1 queues.

Tran et Al. [35] develop a model-free reinforcement learning framework and suggest a new parametrization of the policies of the queueing models. They claim that for smaller and simpler or very complex types of queues their approach can be useful since even for the simpler queue models with analytical solutions the required parameters for the solution are not known in advance for most of the cases. Thus, they emphasize the importance of having a model-free framework to be able solve the queueing models and propose a new policy parametrization scheme. They apply their framework on a simple parallel queueing system with exponential arrival and service times with different load conditions and claim that logarithm-scale parametrization and finite difference based gradient estimator are superior to other options.

Chen [36] studies the infinite horizon M/G/1 Queueing System for production with demand arrival is a Poisson Process and production time of each product is random. The queueing system models an inventory scenario in which the aim is to minimize the long run average expected cost of inventory. Chen proposes a model free reinforcement learning algorithm with a stochastic gradient update rule on policies. The algorithm learns without any a priori knowledge about arrival and service rates. Chen shows that its algorithm's cost converges to the optimal cost at the best rate

possible theoretically.

Jia et al. [37] consider online learning of pricing in a queueing setting. Their aim is to maximize the expected profit by learning the best pricing in a reusable resource queueing setting. The arrival of customers follows a price-dependent Poisson process. The service time for a resource is exponentially distributed. They assume that the inter-arrival and service times have an unknown linear dependence to the price. In this setting they propose a model based online learning algorithm (Batch Linear Confidence Bound) which learns this unknown dependence and finds the best price. They compare their results with the optimal pricing and prove that the cumulative regret is in the order of linear dependence vector times the time horizon which is the least possible theoretically.

Krishnasamy et al. [38] study learning unknown service rate in queues. They consider a multi-armed bandit setting in which a queue has jobs to be scheduled to one of the servers (bandits) and each server has an a priori unknown probability of successfully completing the job in the queue. In this setting the aim is to minimize the queue length. They study different algorithms and their goal is to minimize the regret which is the expected difference between the algorithm's queue length versus the optimal queue length. They propose model free algorithms which ensures the regret increases no more than logarithmically both in the early stage and in the late stage of the queueing system. They use Upper Confidence Bound and Thompson sampling approaches on estimation, exploration and they create a policy based on the estimated values of μ . They prove their results for more than one job classes.

Zhong et al. [39] consider a scheduling and queueing problem in which the controller tries to assign customers in queues to the servers in a multi-class many server environment with abandonment. Customers from different classes have different patience times in which after the wait exceeds the patience they leave. The objective is to minimize the average long run cost of holding and abandonment where the model parameters such as arrival, service and abandonment rates are unknown. They ap-

ply an algorithm which first learns the parameters and then applies a known optimal policy to the system. They balance the exploration and exploitation of phases of the algorithm to achieve good results. They prove that their algorithm reaches a regret upper bound of $O(\log T)$.

Shah et al. [40] consider reinforcement learning with unbounded state space. They claim that though reinforcement learning approaches work well for the finite or countable state space cases there is not enough work for the cases with the unbounded state space. They draw inspiration from queueing systems and control theory and propose the notion of stability of the queueing system as a performance metric for the systems with unbounded state spaces. They investigate the characteristics of a policy which ensures a stable system and propose an adaptive learning algorithm which finds a policy that ensures a stable system. The algorithm is a model free algorithm called Sparse Sampling Monte Carlo Oracle which is a mixture of look ahead (temporal difference) and monte carlo methods.

A summary of the studies and the approaches used by the studies represented in Table 2.1 below. As seen, there are various applications of RL methods regarding queueing problems. The trends and tendencies on the literature show us that the studies on the first category tend to use model-free approaches more than the model based approaches. This tendency is reasonable due to the complex network structures they focus on. Since using a model free approach is easier to implement and less demanding on modelling. Additionally, there is a tendency to use policy based methods such as Policy Gradient and Policy Iteration. For the studies on the second category which our study also belongs to, we can see the same tendency of using model-free methods. This tendency shows that the model-free approach is useful for both cases. Again there is a dominance of policy based methods for the type of algorithm that is used. Only the paper of Shah et al. uses an approach that mixes Q-Learning and MonteCarlo methods. In our literature review, we did not encounter the usage of the Q-Learning algorithms on the queueing domain. We also observed that most of the studies propose an online algorithm which has the functionality to adapt to changing

Table 2.1. Summary of the studies and their approaches in the literature.

Article	Category	RL Category	Application Area	RL-ML Domain
Liu, B., Xie, Q., & Modiano, E. (2019) [24]	1	ModelBased	Serv.All.&Routing	Problem Specific RL
Dai, J. G., & Gluzman, M. (2022) [25]	1	ModelBased	Criss-Cross Networks	PolicyGradientMethods
Bachl, M., Fabini, J., & Zseby, T. (2020) [26]	1	ModelFree	Computer Networks	DeepRL
Raeis, M., Tizghadam, A., & Leon-Garcia, A. (2021) [27]	1	ModelFree	Computer Networks	PolicyGradientMethods
Xu, Z. et al (2018) [28]	1	ModelFree	Traffic Engineering	PolicyGradientMethods
Kim, M., Jaseemuddin, M., & Anpalagan, A.(2021) [29]	1	ModelFree	IoT Networks	DeepQNetwork
Hsu, W. K., Xu, J., Lin, X., & Bell, M. R. (2018) [30]	1	ModelFree	Computer Networks	PolicyIteration
Sentenac, F., Boursier, E., & Perchet, V.(2021) [31]	1	ModelFree	Computer Networks	Multi Armed Bandit Approach
Carner, J., Mestres, A., Alarcón, E., & Cabellos, A. (2017) [32]	2	ModelFree	Jackson Networks	SupervisedLearning
Chen, X., Liu, Y., & Hong, G. (2023) [33]	2	ModelFree	M/GI/1 Queues	PolicyGradientMethods
Chen, X., Liu, Y., & Hong, G. (2023) [34]	2	ModelFree	GI,/GI/1 Queues	Policy Gradient Methods
Tran, T. H., Nguyen, L. M., & Scheinberg, K.(2022) [35]	2	ModelFree	Job Classification (M/M/c)	Policy Gradient Methods
Chen, W. (2019) [36]	2	ModelFree	Inventory Control(M /GI/1)	Policy Gradient Methods
Jia, H., Shi, C., & Shen, S. (2022) [37]	2	ModelBased	M/M/c (Finite Horizon)	Problem Specific RL
Krishnasamy, S., Sen, R., Johari, R., & Shakkottai, S. (2021) [38]	2	ModelFree	M/GI/c Queues	Problem Specific RL
Zhong, Y., Birge, J. R., & Ward, A.(2022) [39]	2	ModelBased	(Mt/M/N+M & GI/GI/N+GI)	Problem Specific RL
Shah, D., Xie, Q., & Xu, Z. (2020) [40]	2	ModelFree	Queue Stabilization	SparseSampling-MonteCarlo-Oracle

environments. As we mentioned in our reasoning to use an online and model free algorithm, the advantages of these methods also show themselves in the literature.

Despite significant advancements, several gaps remain in the literature. The queuing control part of the RL literature focuses on the problems that do not have known optimal solutions. These studies use the RL approach to solve their problems and show that RL methods are effective based on comparing their methods with the other methods in literature. Also, the studies in the literature show that both studies on the first and the second category generally use discounted models and discounting cost structures. This is very common since it provides convergence for the infinite horizon models and the general RL theorems regarding convergence often proved for the discounted versions. Therefore we observe that there is a gap in the literature regarding the topic of solving undiscounted queueing problems with known optimal solutions. Solving a well-known queueing problem with known optimal solution allows us to test the efficiency and effectiveness of the RL algorithms more reliably. We test the effectiveness and efficiency of the Q-Learning algorithm by comparing it with the optimal solution of the MDP model. Additionally we test the Q-Learning on the M/G/1/K version of our problem to see its effectiveness on the non-Markovian settings. We test the model by giving the M/M/1/K MDP solution as a starting point. We then compare this model and the ordinary model to test the effectiveness of embedding the model knowledge to the algorithm. We use the average expected cost per iteration criterion on our study which is an undiscounted cost structure. It is much more suitable for queueing problems and easier to interpret. It is also easier to compare the MDP and RL models' solutions using this structure. The usage of the Q-Learning algorithm and its variants are rare in the literature as we observed. Using the Q-Learning algorithm and its convergence proof we conjecture that Q-Learning algorithm converges to the optimal policy for M/M/1/K problems with the undiscounted costs. Then we numerically show that the algorithm converges to the MDP optimum in some cases and gets very close in others. Moreover, the studies on the literature do not focus on the hyperparameter search for the application of the RL algorithm. There is a lack of comparison on which hyperparameter values and strategies are better and

how much they impact the RL algorithms for queueing problems. We argue that it is a crucial step and directly effects the algorithm's convergence to the optimal policy and convergence rate. We compare 3 different strategies for exploration-exploitation trade-off and try a total 864 different cases with 3 strategies for different problems and report the best performing strategy together with its parameters. This thesis aims to address the gaps explained above by applying Q-learning to M/M/1/K queueing systems, comparing its performance with optimal policies derived from MDP models, and extending the approach to M/G/1/K systems. This research will provide insights into the applicability and efficiency of Q-learning for well-studied problems of queueing control.

In summary, the literature demonstrates the growing importance of reinforcement learning in optimizing queueing systems, with various approaches showing promise. Namely Dai et al. [25] claims that RL approaches now can solve the hardest stochastic problems, Xu et al., Hsu et al. and Chen et al. [28], [30], [33] claims that RL approaches are better than traditional approaches. The advantages of RL approaches on both very complex computer network problems queueing problems direct many researchers to the area. Thus we also believe that there is a need for further exploration of RL methods and their applications to the queueing theory problems. Especially we argue that RL algorithms are useful even for the cases with well-known optimal solutions because they have significant advantages in practice, such as being very flexible and can be continued to use even when the arrival and service patterns change.

2.3. Background on Q-Learning

Reinforcement Learning (RL) is a branch of Machine Learning methods focusing on learning by experience. The way it differentiates from Supervised Learning methods is that in the Supervised Learning scenario the models try to learn the association between the features and targets by using a function or a family of functions which inherently suggest that we have the knowledge of features and the outputs in advance. Reinforcement Learning Models do not know the features nor the outputs and start

learning by experience from the scratch (model free methods) or exploiting a known structure (model based methods). On the other hand it differentiates from Unsupervised models by the aim of the model in which the Unsupervised Models focus to discover a pattern or an association in a given data. Reinforcement Learning methods try to match the given environment with the best course of action. In their fundamental book Richard S. Sutton and Andrew G. Barto defines Reinforcement Learning as “Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.” [19]. Reinforcement Learning borrows many concepts from Markov Decision Processes. To clarify Sutton and Barto, in Reinforcement Learning the environment is modeled with different situations or states it is in, and one can be considered as an agent that observes these states. The agent can take different actions in different states of the environment and every action creates an immediate reward signal. The agent’s goal is not to maximize the current reward but to maximize the long-term sum of rewards. In a stochastic environment, a transition from a state and action pair to another is random. More specifically, the nature of this randomness modeled by the Markovian Property borrowed from the Markov Decision Processes, which dictates that the probability of going to another state depends only on the current state and the taken action. In the stochastic environment, the agent’s goal is to maximize the expected sum of rewards. As briefly summarized, reinforcement learning base its foundation on the Markov Decision Processes and uses the same elements (Decision Epochs, States, Actions, Rewards, Transition Probabilities) to model an environment. The RL domain can be separated into different categorizations such as model-free methods and model based methods or tabular methods and approximate solution methods. A summary of the domain and the related methods that was used in this paper is summarized in Figure 3.2

Our focus in this paper will be on a temporal difference learning method named Q-Learning [41]. The advantage of the Q Learning is that it is a model-free method and provide useful insights regarding every state and action combination in a given model due to its tabular nature. As their category suggest Q-Learning uses a tabular structure and makes an estimation table regarding the long-term reward of states and

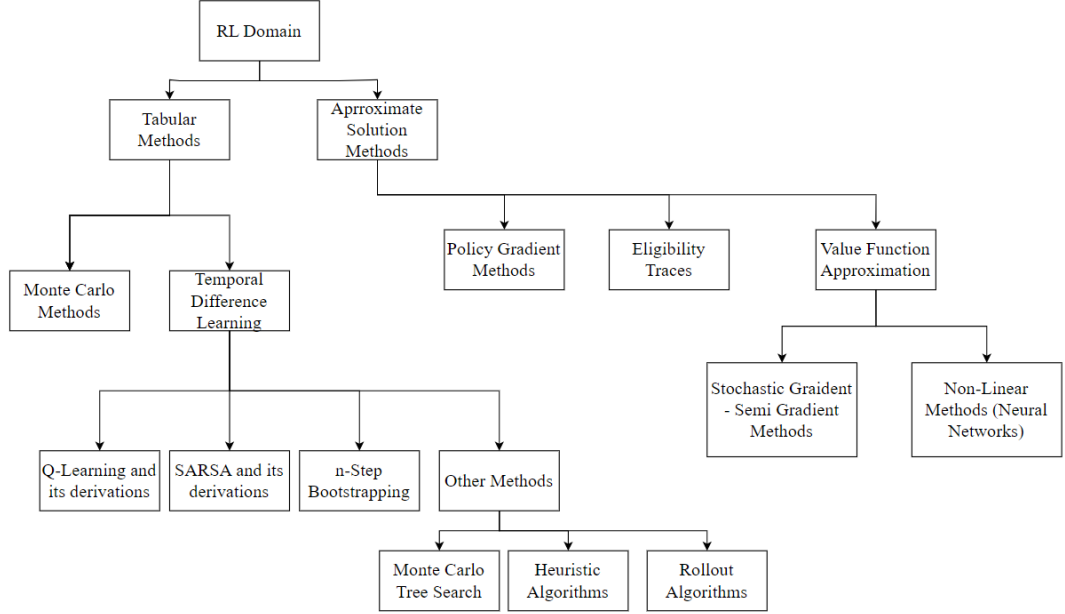


Figure 2.1. Reinforcement Learning Domain Classification

actions combinations. This structure makes it eligible for problems with discrete state and action sets such as ours. Q-Learning algorithm is a temporal difference method. The temporal difference learning methods try to estimate the state-action values for a given policy. To define this concept clearly, definition of the value function is helpful. Given an environment modelled with Markovian Decision Process models where S is the state set, which is defined as

$$S = (s_1, s_2, s_3, \dots, s_t, \dots) \quad (2.1)$$

while the action set A is defined as

$$A = (a_1, a_2, a_3, \dots, a_t, \dots) \quad (2.2)$$

And the reward set R is defined as

$$R = (r_1, r_2, r_3, \dots, r_t, \dots) \quad (2.3)$$

which together forms the fundamental elements of the Markov Decision Process where t is the decision epoch or the time step, s_t is the state is in at the time step t , a_t is the action taken at the time step t and r_t is the reward incurred at the time step t . The

value function for a given policy, denoted as v_π defines the total expected discounted reward (undiscounted when the discount factor is 1) obtained for a given problem provided that the process started from the state s and the policy π is followed until the end of the horizon or for infinite horizon models until infinity. Similarly $q(s, a)_\pi$ is the total expected discounted reward obtained for a given problem provided that the system started from the state s and action a taken at the first decision epoch and after that the policy π is followed. The state value function of an optimal policy is denoted as $q(s, a)_{\pi^*}$. Once we have the $q(s, a)_{\pi^*}$ values for all of the possible states and actions, the only thing left is to pick the action that provides the maximum/minimum $q(s, a)_\pi$ value for the regarding state to determine the optimal policy. The below equations express a clear definition for the value function and the state-action value function. The value function is defined as

$$v_\pi = E_\pi[r_t + \alpha R_{t+1} | S_t = s] \quad (2.4)$$

which can also be expressed as

$$v_\pi = E_\pi[r_t + v_\pi(S_{t+1}) | S_t = s] \quad (2.5)$$

that allows us to write it in its open form as

$$v_\pi = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) [r + \alpha v_\pi(s')] \quad (2.6)$$

Using the open form of value function we can express state-action value function using

$$q(s, a)_\pi = E_\pi[r_t + \alpha v_\pi(S_{t+1}) | S_t = s, A_t = a] \quad (2.7)$$

$$q(s, a)_\pi = \sum_{s'} p(s'|s, a) [r + \alpha v_\pi(s')] \quad (2.8)$$

where $t = \{1, 2, \dots, T\}$ represents time steps or decision epochs, r_t represents the current reward (the reward obtained at time step t), R_t represents the total expected reward starting from time $t+1$ onwards and α represents the discount rate. $v_\pi(S_t)$ defines the total discounted sum of rewards calculated starting from the decision epoch t until the T . ($T \leq \infty$). Q-Learning algorithm is an estimation scheme for the optimum

state-action values. Once these values are estimated one can create a policy by choosing the actions greedily for any given state. The term temporal difference defines the estimation structure. The sample path of a Markovian Decision Process can be denoted as below. Sample path refers to the possible states, actions and incurred rewards for a process starting from $t = 0$ onwards as defined in equation

$$\Omega = (S_0, A_0, R_0, S_1, A_1, R_1 \dots) \quad (2.9)$$

and denoted with Ω . In practice without knowing the model parameters and the transition probability function of the process, we cannot generate it but we can observe a realization of such patterns given a policy, which is represented in equation

$$\omega = (s_0, a_0, r_0, s_1, a_1, r_1 \dots) \quad (2.10)$$

and denoted with ω as realized pattern. This realization means a possible sample ω from the set of all of the sample paths for a given process Ω . Using possible samples such as ω we can estimate the state action value function $q(s, a)_\pi$. Now we can define Q-Learning [41] and explain the meaning behind the term temporal difference. The below equation

$$Q(S_t, A_t) = Q(S_t, A_t)_n + \beta[r + \alpha \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (2.11)$$

defines the estimation procedure and the main mechanism of Q-Learning. It updates its estimate of $q(s, a)_\pi^*$ the optimal state-action value function by using the previous estimate and a difference between the current estimate and the new gained information regarding estimates. As Sutton and Barto explains it [19], the quantity in brackets is a sort of an error, measuring the difference between the estimated value and the better estimate observed as $r + \alpha \max_a Q(S_{t+1}, a)$. This quantity is also called as temporal difference hence the name temporal difference (TD) learning. Again a very concise explanation regarding the explanation of the estimation scheme is by Sutton and Barto as follows, The TD target is an estimate for both reasons: it samples the expected values in and it uses the current estimate instead of the true value. Thus, TD methods combine the sampling of Monte Carlo methods with the bootstrapping of

Dynamic Programming. [19] As they mention we do not know the real $q(s, a)_\pi$ values but we estimate it using data in the form of ω using many sample paths mean that we do a Monte Carlo estimation. However our estimation of the current state S_t is based on the estimate of the next state's value $S_t + 1$. This is very similar to the Value Iteration algorithm in principle [42] which also starts from arbitrary initial values and converges to the correct values by using the value of the next states to calculate the value of the current state. The big difference is in Value Iteration we assume we know the parameter values, while in Q-Learning we estimate the values at the same time we optimize the actions. Q-Learning is a model free and off-policy method. The term model free refers to the fact that it does not estimate the transition probabilities of an assumed MDP model of the environment. The term off-policy refers to the fact that it can learn optimal state value function, by using data created implementing any given policy. To be more precise, any policy which assigns probabilities of selection to all possible actions that are greater than 0 in any state, it can learn the optimal state value function. The only condition is to apply a stochastic policy which gives possibility to each action in each state because the algorithm needs data regarding any possible state-action match to learn. One very popular suitable policy structure is called as ϵ -greedy structure that allows the controller to take different actions from its current policy with probability of ϵ . This structure is actually an exploration strategy which allows the model to explore and estimate the other actions if they are better to choose them. Though this is not the only structure that can be used for exploration as we will mention later. Being an off-policy method is a liberating property since it can use data created by any policy that fits the criteria. This also allows and defines the Q-Learning algorithm as an online learning method. This means it can learn in live interactions as the data is obtained through the modelled process. It can also change the policy that is applied and continue to learn if the environment changes. Those 3 characteristics are the strengths of the method which we argue in this thesis.

3. METHODOLOGY

3.1. Problem Definition and Objective

This study focuses on a problem in which a single server capable of serving in alternative speeds receives jobs that arrive at the server one by one after random inter-arrival times. Arrived jobs are served with a first come, first serve (FCFS) policy. The capacity for the total number of jobs in the system is limited to K . The controller aims to balance the queue length with the cost of alternative serving speeds. A holding cost is used to penalize holding jobs in the system, while a service cost proportional to the service rate is incurred for serving faster or slower. The controller decides the serving speed according to the number of the jobs in the system and sets the rate at each arrival and departure preemptively.

The optimal control problem of switching rates can be analyzed by assuming independent arrival and service rates that follow exponential inter-arrival times with respective rates λ and $\mu_i, i = 1, 2$ where $\mu_1 < \mu_2$. Under the Markovian assumptions the optimal control of the rates can be studied using Markov Decision Process (MDP) or the machine learning (ML) algorithms. Specifically, we will use reinforcement learning (RL) algorithms, a subset of machine learning (ML) that focuses on problems similar to optimal control problems. Our objectives in this thesis are twofold. First, we will compare the efficiency and effectiveness of the RL algorithm's (Q-Learning) variants with the MDP results. Then, we will relax the queueing model by considering non-exponential service rates and using the RL method to solve the problem. Here, we will study the benefits of feeding the MDP results of M/M/1/K problem as an initial solution to the Q-Learning method (MDP-Informed model) in the solution of the relaxed model. We will compare the performances of the ordinary Q-learning method and the MDP-Informed Q-learning method to study the benefits. To achieve our objective, we will develop an MDP model and Q-Learning algorithm to solve the same problem. We will try three different exploration strategies with Q-Learning algorithm. Then,

we will compare them with two different methods for the two objectives. The performance of the Q-Learning models in the M/M/1/K problem will be measured using the value iteration algorithm to calculate the average expected cost per decision value. The performance of the Q-Learning models in M/G/1/K model will be measured using simulations.

3.2. Model Development

In this section, we will develop the MDP model, the RL model, and the simulation environment we use to train the RL model and compare the results of the non-Markovian problem.

3.2.1. MDP Model

Number of jobs in the system is modeled as the state set. $S = \{0, 1, 2, \dots, K\}$ and the average transition rates are the arrival (λ) and service rates (μ_i , $i = 1, 2, \dots, \mu_1 < \mu_2$). As the state set is defined as number of jobs in the system, the next element of our MDP problem is the reward function. It is the sum of holding costs and service costs for the selected rate. Since our reward is defined as a cost our goal is to minimize the average expected cost or long-run expected cost. The transition probabilities of our MDP can be derived using the transition probability matrix and the sojourn rates of the CTMC structure. The transition probabilities and the rates can be easily calculated; however, the derivation of the probabilities will be omitted here as the problem will be solved with the discretization of the system using Bellman equations. The structure mentioned above defines an Infinite Horizon Discrete State Markov Decision process. It can be solved by solving the Bellman equations which is

$$V_{n+1}(X) = hx + \frac{1}{\Lambda} \{ \lambda V_n(X+1) + \min (\mu_i V_n(X-1) + \mu_i c_i) + (\Lambda - \lambda - \mu_i d_i) V_n(X) \} \quad (3.1)$$

where X is the number of jobs in the system, hx is the holding cost, λ is the arrival rate, μ_i is the possible service rates with costs of $\mu_i c_i$, d_i is the decision variable to choose between the service rates where $i = 1, 2$, $d_i \in \{0, 1\}$, Λ is the total of arrival rate and

the maximum service rate, and n represents the n th iteration of the algorithm to solve the Bellman equations. As the new value of the equation represented by $n+1$ and the old value represented by n . The value function V_n is defined in the equation 3.1 takes expectations by multiplying the equations with arrival and service rates. In average the system will have an arrival with probability $\frac{\lambda}{A}$, a departure with probability $\frac{\mu_i}{A}$. The rest of the probability $\frac{A-\lambda+\mu_i d_i}{A}$ is the probability of the system staying with the same number of jobs. The variable d_i is a binary variable representing the selected service rate. The described equations represent the system dynamics on average and minimize the expected average cost in the long run. Since our case represents a M/M/1/K problem the Bellman equations of the state X which defines the solution for all of our state space, starts from 0 and goes up to K . This means our equations have bounds on 0 and K . The first equation which is as below

$$V_{n+1}(0) = \frac{1}{A} \{ \lambda V_n(1) + \min(\mu_i V_n(0) + \mu_i c_i) + (A - \lambda - \mu_i d_i) V_n(0) \} \quad (3.2)$$

explains the dynamics of bound 0 whereas the below equation

$$V_{n+1}(K) = hK + \frac{1}{A} \{ \lambda V_n(K) + \min(\mu_i V_n(K-1) + \mu_i c_i) + (A - \lambda - \mu_i d_i) V_n(K) \} \quad (3.3)$$

explains the dynamics of bound K . Finally the equation

$$V_{n+1}(X) = hX + \frac{1}{A} \{ \lambda V_n(X+1) + \min(\mu_i V_n(X-1) + \mu_i c_i) + (A - \lambda - \mu_i d_i) V_n(X) \} \quad (3.4)$$

explains all the cases that are not in bounds. The Bellman equations take 3 possible cases into account. Possible arrival case indicating an increase in the number of people in the system, possible departure case indicating a decrease in the number of people in the system and possible no change case indicating a constant number of people in the system. In the bounds of 0 and K the possibilities are different. At the bound 0 there can be no departure but the number of people increases with the rate λ or stays the same with the rates μ and $A - \lambda - \mu$. At the bound K there can be no increase since the capacity is full but a possible arrival before departure means a that number of people in the system is still K so, the number of people stays the same with the rate λ and $A - \lambda - \mu$ or decreases with the rate μ . The above equations model the problem

as a discretization of the continuous time system by using the expected average cost of the MDP model. The optimum policy can be calculated with various methods such as Policy Iteration, Value Iteration or Linear Programming. In our case we used Value Iteration to obtain the optimum policy and compared it with our reinforcement learning model's result. We used the second order differences which will be explained in more detail in the next section to check the convergence of the algorithm. The threshold used for the convergence is 10^{-7} . And the first difference is explained by the equation

$$d_1 = V(X)_{n+1} - V(X)_n \quad (3.5)$$

while the second difference is explained by the equation

$$d_2 = V(X)_n - V(X)_{n-1} \quad (3.6)$$

finally the equation

$$\text{Stop once } \max(d_1 - d_2) \leq 10^{-7} \quad (3.7)$$

explains the the criteria of second order differences where $n+1$ and n represents the iteration numbers of the value iteration algorithm. As the iterations progress, the difference between the calculated values gets smaller.

For the M/G/1/K extension of the problem we compared the results on the simulation to check the performance of the RL model. Our aim was not to find a specific optimum solution. Thus omitted the necessary derivations for the optimal solution of the extension case.

3.2.2. The Q Learning Algorithm, Convergence and Convergence Criterion

In this section, we first discuss the working mechanism of the Q-Learning algorithm and show its similarities to the value iteration. Then, we present the proof of convergence of Q-Learning to the optimal state- action values. Additionally we show our reasons regarding the validity of the proof in our problem. Then we show the

convergence criteria of the value iteration algorithm and develop a convergence criteria for Q-Learning using the similarities between the Q-Learning algorithm and the value iteration algorithm. As discussed in the background section the key equation of the Q-Learning algorithm is

$$Q(S_t, A_t) = Q(S_t, A_t) + \beta[r + \alpha \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (3.8)$$

repeated as above. As the equation 3.8 shows, the Q-Learning algorithm uses a very similar mechanism to the value iteration algorithm but uses state-action values instead of state-based value functions. Starting with arbitrary values, the Q-Learning algorithm creates a table for the optimal state-action values. Matrix shown in equation

$$\mathbf{Q}(\mathbf{s}, \mathbf{a}) = \begin{pmatrix} Q(s=0, a=0) & Q(s=0, a=1) & \cdots & Q(s=0, a=n) \\ Q(s=1, a=0) & Q(s=1, a=1) & \cdots & Q(s=1, a=n) \\ Q(s=2, a=0) & Q(s=2, a=1) & \cdots & Q(s=2, a=n) \\ Q(s=3, a=0) & Q(s=3, a=1) & \cdots & Q(s=3, a=n) \\ Q(s=4, a=0) & Q(s=4, a=1) & \cdots & Q(s=4, a=n) \\ Q(s=5, a=0) & Q(s=5, a=1) & \cdots & Q(s=5, a=n) \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} \quad (3.9)$$

is an example of such a table and it represents the $Q(S_t, A_t)$ values in the equation. Similar to the iterations of value iteration this table starts with arbitrary values and updated in each iteration of the algorithm using the equation 3.8 for every state-action pair. The new estimate is obtained by the existing values in the current estimate and a fraction of the term in the brackets. The term in the brackets consists of the difference between the new estimate and the old estimate. This difference multiplied with the learning rate β and added to the current estimate $Q(S_t, A_t)$ to create the new estimate for the corresponding state and action.

This term is called the temporal difference. It uses the dynamic programming logic here. The total expected cost of a policy is the current cost plus the expectation of the costs from the current time step to the end. Here r is the current cost, α is the discount which is 1 and ineffective in our case, and the term $\max_a Q(S_{t+1}, a)$ is the best action value of the next state. Here, it uses the value of the following state as an estimate of the expectation of the total expected cost. The term together $r + \alpha \max_a Q(S_{t+1}, a)$ serves as an estimation of the total expected cost and that value is subtracted from the value of the current estimation $Q(S_t, A_t)$. Now, starting with arbitrary values such as 0, for every state and action value, we record the values of those actions as we observed them in the system or in the simulation in our case. Using them, at first, we only have the current reward as the next step's value is unknown, and any other $Q(s, a)$ values are unknown. However, as the system progresses and starts to have data regarding the following states, the values begin to form and create the estimates. It is very similar to the value iteration in this sense. The main difference is that value iteration knows the real values of the parameters and starts to converge to the true values as iterations progress. On the other hand Q-Learning algorithm starts with observed estimates and updates its estimates as the iterations progress. It makes the estimation and optimization at the same time. Another difference is that the state action values are based on a policy as the notation on the background section $q(s, a)_\pi$ values identify the values of the state s when the first action taken is a and the other actions are taken according to the policy π from the second action onward. Thus, the Q-Learning algorithm makes the estimation based on a dataset created by a policy π . However it is an off-policy method which means it can learn the optimum state-action values from any dataset created by any policy that provides the conditions mentioned in the background. The advantage is that once the algorithm learns the optimal state-action values, choosing the best action in each state gives us the optimal policy.

The convergence of the Q-Learning algorithm to the optimal policy has been showed by Watkins and Dayan in [43]. Under the specific conditions for learning rates and bounded rewards, as the number of episodes goes to infinity the algorithm finds

the optimal state action values with probability 1. The theorem from the paper is as below. The implicit assumption of the Markovian Property is necessary for the below theorem. Giving n represents the number of the episode and $(s, a)^i$ represents the i th time the action a is taken on the state s , the theorem expresses that with an appropriate decaying learning rate of β as the number of episodes goes to infinity the Q-Learning algorithm will find the optimal state action values with the probability 1. [Theorem 2.1 Convergence of Q-Learning] [43] Given bounded rewards r_n , learning rates $0 < \beta_n < 1$, and the conditions:

$$\sum_{i=1}^{\infty} \beta_n(s, a)^i = \infty \quad \text{and} \quad \sum_{i=1}^{\infty} [\beta_n(s, a)^i]^2 < \infty, \forall s, a \quad (3.10)$$

then:

$$q(s, a)_\pi \rightarrow q(s, a)_{\pi^*} \quad \text{as} \quad n \rightarrow \infty \quad \text{with probability 1.}$$

The paper first offers this proof of convergence regarding theorem 2.1 [43] for the case of discounted rewards. Then, it expands it to the case of undiscounted rewards with the existence of an absorbing state. Though it does not mention the specific extension to the undiscounted rewards case without the existence of absorbing states, we claim that the capacity restriction on our case can play the role of absorbing states, which then ensures the convergence. We justify this by comparing the convergence conditions to our case as below.

[Conjecture 2.1 Convergence of Q-Learning in M/M/1/K Queuing Systems] Given an M/M/1/K queuing system, where:

- (i) The state space is finite due to the finite capacity K .
- (ii) Rewards are bounded.

(iii) Learning rates $0 < \beta_n < 1$, satisfy:

$$\sum_{i=1}^{\infty} \beta_n(s, a)^i = \infty \quad \text{and} \quad \sum_{i=1}^{\infty} [\beta_n(s, a)^i]^2 < \infty, \forall s, a \quad (3.11)$$

then

$$q(s, a)_{\pi} \rightarrow q(s, a)_{\pi^*} \quad \text{as} \quad n \rightarrow \infty \quad \text{with probability 1.}$$

In our conjecture 2.1, we claim that given our conditions, the convergence proof for Q-learning provided by Watkins and Dayan (1992) [43] applies to our M/M/1/K queuing system for the following reasons.

Finite State Space: The state space $\{0, 1, \dots, K\}$ is finite, and the underlying Markov Chain is irreducible, ensuring that all states are revisited and updated.

Bounded Rewards: In our case we define our rewards(costs) of holding the jobs and the serving the jobs to be bounded. Our holding cost proportional to the number of jobs in the system which can be K at the most, and a positive constant smaller than the infinity. Our serving cost is proportional to the service rates and service costs which are all set as positive values smaller than infinity.

Bounded Value Functions: For the undiscounted case, the convergence proof requires the existence of absorbing states. The reason for that is explained by the writers as A process with absorbing goal states has one or more states which are bound in the end to trap the agent. This ultimate certainty of being trapped plays the role of $\gamma < 1$ played in the earlier proof in ensuring that the value of state x under any policy π , $V^{\pi}(x)$, is bounded, and that lemma B.1 holds, i.e., that the difference between considering infinite and finite (s) numbers of actions tends to 0 as $s \rightarrow \infty$. [43] This emphasizes that the requirement for the absorbing state is to make sure that the value of any state under any policy $V^{\pi}(x)$ is bounded. Even though states 0 and K are not absorbing, the value functions are bounded because the system has a finite

number of states, ensuring that no state can have an unbounded value. Additionally, from another perspective, we know that when an absorbing state exists in an infinite Markov Chain, the process is going to have a finite time, although the time of the absorbance is uncertain. On the other hand, an irreducible Markov Chain, even when $|S| < \infty$, does not satisfy the finite time property. Yet from the MDP convergence, we know that at the limit, an irreducible Markov Chain converges to the π^* . Here, we assume that the same will work out since the number of possible paths is less than infinity.

Most of the same conditions hold for the M/G/1/K extension of the queueing systems. However, the implicit condition of being in a Markovian environment prevents us to state that the algorithm converges to the optimal state action values $q(s, a)_{\pi^*}$ for the M/G/1/K case. Nevertheless providing the convergence conditions for an M/M/1/K model as mentioned above we believe that the algorithm will converge to fixed state action values $q(s, a)_{\pi}$. This gives us the opportunity to test the algorithm performance and convergence on these models using numerical experiments.

After justifying that the algorithm converges to optimum policy, we now need to develop a criterion for the termination of the algorithm. The similarities between the Value Iteration algorithm and the Q-Learning algorithm inspire us to determine a termination-convergence criterion. Before explaining our convergence criterion, we need to explain the convergence criterion used in Value Iteration. In our implementation of the Value Iteration algorithm, since our problem is undiscounted, we use the average expected cost criterion as our long-term cost criteria, and to determine that the algorithm converged, we measure the second-order differences. The first difference is expressed in equation

$$d_1 = V(X)_{n+1} - V(X)_n \quad (3.12)$$

while the second difference expressed in equation

$$d_2 = V(X)_n - V(X)_{n-1} \quad (3.13)$$

then finally the second-order differences approach to zero as expressed in

$$\text{As } n \rightarrow \infty, d_1 - d_2 \rightarrow 0 \quad (3.14)$$

where n indicates the number of the iteration of the Value Iteration algorithm, the difference between the values calculated on the iterations $n + 1$ and n denotes the first difference (d_1) while the difference between the values calculated on iterations n and $n - 1$ denotes the second difference (d_2). As shown in summarizing equations of 3.12, 3.13 and 3.14, as the iterations continue the difference between d_1 and d_2 approaches 0. This criterion is based on the fact that since in the discounted Value Iteration the difference of the values approaches 0 [42] for the undiscounted case, the difference of the values should approach to constant value. Due to the absence of a discount factor the values continue to increase but the increase per iteration becomes stabilized since the MDP structure assumes a steady state behavior. Thus the difference between the differences should approach to 0 in this situation.

This constant increase represented by d_1 and d_2 is also named as average expected cost per iteration. This criterion is used frequently in queueing control because it can be thought of the cost per decision epoch. Cost per decision is a logical metric in problems like queueing control where the system evolves based on the frequent decisions made by the agent. The frequent decisions are also the reason for using the undiscounted costs since the time between decisions are short and discounting is not much realistic. Thus, we used undiscounted costs.

Inspired by the similarities between the value iteration algorithm and the Q-Learning algorithm we, decided to use the same convergence criteria. The only difference is that while on the value iteration the algorithm has the real values, Q-Learning algorithm checks the second order difference between its estimates. The first difference of estimated values is expressed in equation

$$\tilde{d}_1 = \tilde{v}(s)_\pi^{n+1} - \tilde{v}(s)_\pi^n \quad (3.15)$$

while the second difference of estimated values expressed in equation

$$\tilde{d}_2 = \tilde{v}(s)_\pi^n - \tilde{v}(s)_\pi^{n-1} \quad (3.16)$$

then finally the second-order differences of the estimated values approach to zero as

$$\text{As } n \rightarrow \infty, \tilde{d}_1 - \tilde{d}_2 \rightarrow 0 \quad (3.17)$$

where n implies the number of iterations, $\tilde{d}_1$ and $\tilde{d}_2$ are the differences between the estimated value functions of $\tilde{v}(s)_\pi^{n+1}$, $\tilde{v}(s)_\pi^n$ and $\tilde{v}(s)_\pi^{n-1}$. The estimated value functions is obtained by the estimation table of the Q-Learning algorithm by selecting the optimum action's value at that iteration. As one can obtain the $v(s)_\pi$ values from the $q(s, a)_\pi$ values which can be derived from the equations given in the background [19]. In implementation, both for the Q-Learning algorithm and value iteration we use maximum of difference vector as our convergence value. Since if even the maximum of the vector is smaller than the determined convergence value, we can say that the algorithm is converged.

After the explaining the guarantees for the convergence of the algorithm and a criterion to check the assumed the convergence is established, we can focus on the hyperparameters and preferences about the algorithm that affects the convergence rate and the performance.

3.2.3. Convergence of The Q Learning Algorithm and Hyperparameter Search

The conditions mentioned on the proof [41] also reveal the topics we should focus on for the algorithm's convergence. These are the learning rate and exploration-exploitation balance. In our notation, β is our learning rate, which decides how much the older estimation of the state action values should be updated. We made different experiments for the values of β , but our main decision was to use the approach of Sutton and Barto [19]. Their approach was to use one divided by the number of visits

to the state action values as their corresponding learning rate. They prove that this is equivalent to taking an average of all the values that are calculated throughout the visits to a certain state-action pair. This approach gives the same weight to all updates that occur in all visits. On the other hand, using a constant parameter $\beta \in [0, 1]$ follows an approach similar to the exponential smoothing methods in prediction. As mentioned in the proof, the learning rate should be appropriately decayed for the constant rate case. On the other hand using the approach of Sutton and Barto [19] ensures a decaying rate as the number of visits increases throughout the episodes. This decision ensures that the learning rate is appropriately aligned. This usage of the learning rate also provides a different learning rate for each state-action pair and makes sure that all states and actions have been visited sufficiently enough, together with our convergence criterion. Since the learning rate does not decay sufficiently for the less visited states, the estimations regarding these states will not be stabilized and they create higher difference values. Thus, this usage of the learning rate and our explained convergence criterion work together to create enough visits to all states and a good solution. The second topic is the balance of exploration and exploitation. This is crucial for the part which necessitates infinitely many visits of state and actions. Though in practice one can never provide infinite number of visits, visiting each state and action in a balanced manner ensures better estimations. Exploration is necessary to ensure that the agent tries out all possible actions in all states, preventing the algorithm from getting stuck in sub-optimal regions of the state space. Accurate Q-value estimates require sufficient exploration to gather information about the rewards and transitions for all state-action pairs. If the agent exploits too early, it might miss out on better actions that were not adequately explored, leading to sub-optimal policies. Yet too much exploration can slow down convergence because the agent spends too much time trying sub-optimal actions and cannot learn the optimal actions well enough. However, too much exploitation can lead to premature convergence to sub-optimal policies because the agent does not explore sufficiently. A well-balanced approach ensures that the agent gathers enough information early on (through exploration) and gradually shifts towards optimal performance (through exploitation), leading to reliable and efficient convergence. Additional to the proof by Watkins and Dayan (1992) [43] there are papers which inves-

tigate the role of exploration and exploitation in reinforcement learning domain and its relation to the convergence of the learning algorithm such as [20–22], [44,45]. Based on the reasons above, the main focus of this paper regarding hyperparameter search is on exploration and exploitation strategies together with the hyperparameters needed to use these strategies. The exploration-exploitation trade-off is typically managed using strategies like using ϵ -Greedy policies, Upper Confidence Bound (UCB) [19], [46,47] and Boltzman exploration. [19], [46], [48] There are different approaches as well, but in our review of the literature, we saw that these 3 approaches are commonly used, easy to implement, and are different from each other with different advantages and disadvantages, which motivated us to focus on these methods.

Before diving into the exploration strategies, let us briefly discuss another hyperparameter and explain our preferences. The episode length is the parameter. It may have a crucial effect on the learning process. Based on the intensity ρ of the queueing model, as the number K increases, it becomes harder to reach certain states near the end of the bound. In an $M/M/1$ queue we can calculate the probability of a system going to a particular state given the intensity ρ . For example with $\rho = 0.75$ the probability of going to state 10 or having 10 people in the system is $(0.75)(1-0.75)^{10} = 7.15255737e-7$. As the probability is very low, it is hard for the model to obtain data for that kind of state, and here, the run lengths become more critical. The increase in the number of episodes may not help at this point or may lead to the necessity of too much episodes before convergence. As the run length increases, the model has the chance to observe states near the bound, but increasing the number of episodes does not necessarily increase that chance. Using simulations with smaller lengths, even with the help of many simulation runs, may not reach the rare cases of having many people on the system since they all start with 0 customers. One quick solution to this problem may be using simulations that start with different numbers of people in the system or using very long simulations. However, one should also consider that increasing simulation lengths also increases the run time and computational burden. In our experiments, we observed the effects of increasing episode length. Using simulations to create data makes creating data for rare cases easier. Also, having the capacity constraint K makes

managing the state space easier and reduces the number of rare events. However, for an online training case which one trains the model in real life, the model has limited data, there are various methods such as experience replay [49], prioritized experience replay, which is an extension to improve learning events [23], importance sampling, weighted importance sampling [19] which helps to adjust the weight of the rare events and so on. We direct the reader to the literature and the cited sources here for further research on the topic. Now, we will briefly explain the exploration strategies used.

3.2.3.1. Boltzman Exploration Strategy. The first exploration strategy we will examine is the Boltzman exploration strategy. This strategy creates a probability of selection for each action. The probabilities are calculated based on a Boltzman (softmax) function methodology with a little twist. The formula below is for the minimization case. One can easily convert it to be used for the maximization case.

The probability of selecting action a in state s using Boltzmann exploration for minimization is given as

$$P(a|s) = \frac{e^{-Q(s,a)/\tau}}{\sum_{a'} e^{-Q(s,a')/\tau}} \quad (3.18)$$

where $(P(a|s))$ is the probability of selecting action a given state s and a' represents all possible actions other than a for the state s , $Q(s, a)$ is the estimated cost of action a in state s and τ is the temperature parameter that controls the degree of exploration.

As shown in the equation 3.18 τ is the hyperparameter that determines the action selection behavior. Figure 3.2 demonstrates the behavior of the Boltzman strategy. Given seven actions with the corresponding costs of [10.0, 9.5, 9.0, 7.0, 5.0, 3.0, 1.0] one can easily see that the best action to minimize the costs is action 7. For different τ values of [0.1, 1.0, 3.0, 5.0, 10.0, 20.0, 25.0, 50.0], the figure shows the probability distributions of the action selections under different τ values. As the τ value decreases, the probability of selecting the best action increases and the distribution gets steeper to-

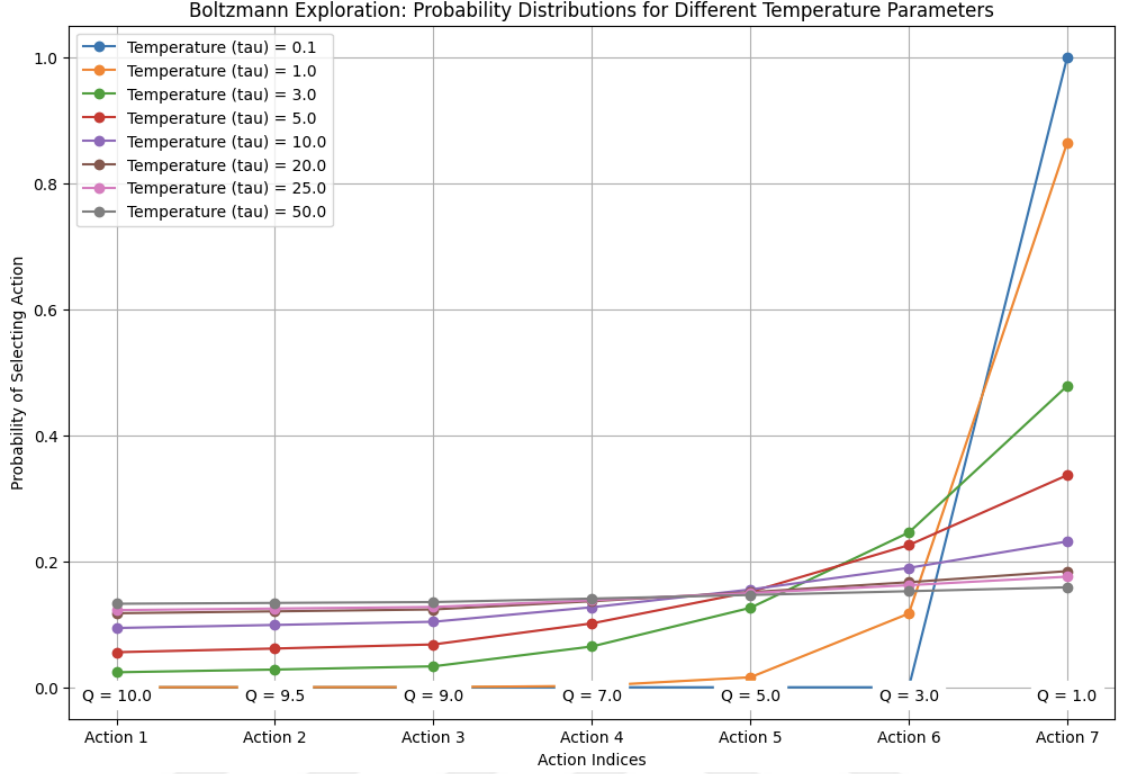


Figure 3.1. Probabilities of selecting actions for different τ values

wards the greedy choice. As τ increases the distribution becomes smoother and starts to behave like a uniform distribution at the extreme case ($\tau = 50$ case). Therefore, when using the Boltzmann strategy one should start with a higher temperature (τ) values to provide exploration at the beginning and then decrease the temperature towards the end to provide exploitation and convergence. Using a decaying temperature rate is a logical strategy to follow when applying this strategy. In our case, we experimented with different beginning values and decaying rates to examine the algorithm's performance and convergence rate. Additionally, in our numerical experiments, we saw that especially on harder problems (problems where the cost of service rates and holding costs are closer to each other, which makes it difficult to decide), the $Q(s, a)$ values that the model learn throughout the iterations are really close to each other. Thus we used smaller temperatures rather than big ones since the costs are already closer to each other. Since the closer values under the same temperature already favor more exploration. One can also check the [48] for a comparison of the Boltzman strategy in Q-Learning with other popular strategies and check [50] for an improvement proposal

of the Boltzman strategy. To avoid the computational intensity and to be able to compare the more frequently used version of the strategy, we decided to use the vanilla version.

3.2.3.2. The Upper Confidence Bound Strategy. The Upper Confidence Bound (UCB) strategy is an exploration technique used in multi-armed bandit problems and reinforcement learning to balance exploration and exploitation [19]. The main logic of the strategy is to explore the less visited states. The less visited state action pairs have higher variance on their estimates by the nature of the estimation. Using this logic, the strategy calculates a value using the current time step and the number of times an action is selected in that state. For the actions that are chosen less, this calculated expression is higher. As the time step moves forward, the nominator, which includes the time step, always gets bigger; however, if an action is selected less frequently, the denominator increases slowly. This increases the expression in the square root, and as the expression increases, it creates more advantage for that action to be selected by making its value smaller. The term c determines how much of an advantage we will give to the less explored actions, meaning it determines the degree of exploration. It is our hyperparameter for this strategy. The formula of the action selection is as follows. Different from the other two strategies this strategy does not create a probability of selection but directly gives the action to be picked based on the formula. The Upper Confidence Bound (UCB) strategy for minimization selects the action a_t in state s_t given that the current time step is t , which minimizes the following

$$a_t = \arg \min_a \left(Q(s_t, a_t) - c \sqrt{\frac{\ln T_t}{N(s_t, a_t)}} \right) \quad (3.19)$$

where $Q(s, a)$ is the estimated cost (Q-value) of taking action a in state s at the time t , c is a tunable parameter that controls the degree of exploration, T is total visits to a given state at the time t , $N(s_t, a_t)$ is the number of times action a has been selected in state s at the time step t . The Upper Confidence Bound (UCB) strategy for minimization selects the action a_t in state s_t given that the current time step is t , which minimizes the above equation 3.19.

3.2.3.3. ϵ -Greedy Strategy. The ϵ -greedy strategy is the easiest one to implement and also easiest to grasp among our strategies. We used the implementation of Sutton and Barto [19] for our problem. Given that $\epsilon \in [0, 1]$, the strategy is to select the greedy action with the probability of $(1 - \epsilon + \frac{\epsilon}{|A(s)|})$ while selecting other actions with a probability of $\frac{\epsilon}{|A(s)|}$. This means it directly gives a probability to be selected to non-greedy actions. One can start with higher ϵ and decay it throughout the iterations. The ϵ itself is the hyperparameter here. In our experiments we tried constant and decaying ϵ values with different decaying rates. The following expressions

$$P(a^*|s) = (1 - \epsilon + \frac{\epsilon}{|A(s)|}) \quad (3.20)$$

$$P(a|s) = \frac{\epsilon}{|A(s)|} \quad (3.21)$$

where $P(a^*|s)$ is the probability of selecting the greedy action, $P(a|s)$ is the probability of selecting the other actions and $A(s)$ is the state of the possible action in the state s and $|A(s)|$ is the cardinality of the set are representing the selection probability of the greedy and the non-greedy actions.

3.2.4. Q-Learning Algorithm

In this section, we have designed our Q-Learning model for the problem. Since Q-Learning algorithm is an online and off-policy model we developed a discrete event simulation to train our model. We used exponential inter-arrival and service times for the M/M/1/K queueing environment. The simulation system generates an arrival, and then the customer who arrives goes to the server to get her service. She may go directly in if the server is empty or if she must wait in the queue. In a first come, first serve (FCFS) manner, the customers have their services. The controller makes decisions according to the policy at hand, both when an arrival occurs and a service finishes. The policy determines the rate at which the service should work for each state of the system. The state is the number of jobs in the system, as in the MDP model. Our design includes a complexity which when the controller makes a decision after an

arrival, and the server is busy. In this case, we adopt a preemptive service policy. The server resets the service process and starts over with the new rate. A similar design is used for the M/G/1/K extension. Since Q-Learning is an online algorithm, it learns as the data starts to flow. Our simulation model creates a Q-Learning object. Q-Learning object includes a controlling policy, an action selection mechanism based on the exploration strategy and an estimation table to estimate the optimal state action values. After the starting of the simulation starting from time step 2 (decision 2) Q-Learning algorithm starts to estimate the parameters. Imitating the online approach, it learns after each decision point (arrivals or departures) in the simulation. As mentioned earlier we used three different strategies for exploration. The three strategies also shape the action selection procedure using the hyperparameter values. Additionally, learning rate and episode length are important decisions. However, we explained our learning rate selection as the number of visits earlier. We tried 16 different values of hyperparameters for three different strategies Boltzman, ϵ -Greedy and UCB. We selected these three strategies because they are well-known and easy to implement as studied in the literature. [18–22], [30], [37, 38]. The general structure of the algorithm is represented for all three the exploration methods. The equation

$$Q(S_t, A_t) = Q(S_t, A_t) + \beta[r + \min_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (3.22)$$

is the key equation of Q-Learning Algorithm for our problem. Since we use an undiscounted structure the α is omitted and the equation is set to minimize the cost. The algorithm estimates the state-action value at the time step t which is $Q(s_t, a_t)$ by using the minimum cost of the next step $\min_a Q(S_{t+1}, a)$. Therefore we need to know the s_{t+1} to calculate the estimation. Considering this, the algorithm must work backwards, only after observing s_{t+1} we can update the estimate of $Q(s_t, a_t)$ value. Now we will define the functions and variables which we use on our implementation.

For each version of the algorithm, as we presented below we initialize our estimation table $Q(s_0, a_0)$ with zeros as the following

$$\mathbf{Q}_{(\mathbf{s}_0, \mathbf{a}_0)} = \begin{pmatrix} Q(s=0, \mu_i = \mu_1) = 0 & Q(s=0, \mu_i = \mu_2) = 0 \\ Q(s=1, \mu_i = \mu_1) = 0 & Q(s=1, \mu_i = \mu_2) = 0 \\ Q(s=2, \mu_i = \mu_1) = 0 & Q(s=2, \mu_i = \mu_2) = 0 \\ \vdots & \vdots \end{pmatrix} \quad (3.23)$$

and visit count table $N(s_0, a_0)$ as

$$\mathbf{N}_{(\mathbf{s}_0, \mathbf{a}_0)} = \begin{pmatrix} N(s=0, \mu_i = \mu_1) = 0 & N(s=0, \mu_i = \mu_2) = 0 \\ N(s=1, \mu_i = \mu_1) = 0 & N(s=1, \mu_i = \mu_2) = 0 \\ N(s=2, \mu_i = \mu_1) = 0 & N(s=2, \mu_i = \mu_2) = 0 \\ \vdots & \vdots \end{pmatrix} \quad (3.24)$$

with 0 visits to any state action path. We initialize state-action path set and rewards set, as empty sets to record the values. We determine β , determine the discount $\alpha = 1$ as our probability is undiscounted. We determine the holding cost per jobs as h , determine the possible actions (service rates) a_1 and a_2 with their service costs c_1 and c_2 . Arrays in 3.23 and 3.24 shows our estimation and visit count tables. The equation

$$State - ActionPath = [(s_1, a_1), (s_2, a_2), (s_3, a_3) \dots] \quad (3.25)$$

expresses the set of state action path which we record using our simulation after each decision epoch. The equation

$$Rewards = [r_1, r_2, r_3 \dots] \quad (3.26)$$

represents the rewards which we calculate and record as the simulation progresses. The equation

$$r_t = s_t h + a_t c_1 \quad (3.27)$$

represent the calculation of the reward in each step.

We should mention an important remark about calculating the rewards(costs). The cost calculations are based on time since the process itself is continuous. For the holding costs of the jobs in the system, we calculate the time that the system has that many jobs, and for the service cost, we calculate the time of the service. Then, for the holding costs, we directly multiply the time by the cost, and for the service costs, we multiply the time by the cost when the service finishes. Here the preemptive structure should be carefully aligned so that no double countings occur in service cost calculations. These calculated costs are the r_t , which are the current or immediate costs. To obtain our cost metric in the simulations, which we will explain in the following sections, we divide the total cost by the simulation time to obtain a unit cost measure.

The State-Action Path and Rewards set are initialized as empty sets. State-Action Path records the state-action pairs and the rewards set records rewards, in our case costs. For each different strategy determine the actions as per below equations.

Boltzman Strategy: Determine a τ value at the beginning of each simulation. Calculate the probability of selecting each action. The equation for probability selection is

$$P(a_t|s_t) = \frac{e^{-Q(s_t, a_t)_n/\tau}}{\sum_{a'_t} e^{-Q(s_t, a'_t)_n/\tau}} \quad (3.28)$$

where $P(a_t|s_t)$ is the probability of selecting action a_t given state s_t and a'_t represents all possible actions other than a for the state s .

ϵ -Greedy Strategy: Determine an $\epsilon \in [0, 1]$ probability at the beginning of each simulation. Calculate the probability of selecting each action. The equations for probability selection is

$$P(a^*|s_t) = (1 - \epsilon + \frac{\epsilon}{2}) \quad (3.29)$$

$$P(a|s) = \frac{\epsilon}{2} \quad (3.30)$$

where a^* is the action that has the smaller estimate in the $Q(s_t, a_t)$ table and the a is the other action.

UCB Strategy: Determine a c value at the beginning of each simulation. The equation for probability selection is

$$a_t = \arg \min_a \left(Q(s_t, a_t) - c \sqrt{\frac{\ln T_t}{N(s_t, a_t)}} \right) \quad (3.31)$$

where T is total visits to a given state at the time t which is $V(s_t, a_1) + V(s_t, a_2)$, $N(s_t, a_t)$ is the number of times action a has been selected in state s at the time step t which $V(s_t, a_t)$. Equations 3.26, 3.27, 3.28 and 3.29 summarize the exploration calculations in each of the different exploration strategy implementations.

Algorithm 1 Q-Learning

```

1: Initialize:
2: Initialize the tables and variables mentioned above. Determine the exploration
   strategy
3: while until a termination condition do
4:   Determine the hyperparameter value of the exploration strategy
5:   Run a simulation for every episode to create data
6:   while until termination of the simulation do
7:     Determine  $s_1, a_1$  based on the strategy using (3.26, 3.27 – 28, 3.29)  $t = 1$  and
        $r_1$  using 3.24 record the  $s_1, a_1$  into State-ActionPath set,  $r_1$  into Rewards set
8:     while simulation is running do
       each decision epoch starting at time step  $t \geq 2$  Read  $s_{t-1}, a_{t-1}$  from State-
       Action Path Determine  $s_t, a_t$  based on the strategy and determine  $r_t$  using
       relevant equations. Record the  $s_t, a_t$  pair into State-Action Path set,  $r_t$  into
       Rewards set. Calculate  $\min_a Q(s_t, a)$  using the values from  $Q(s_t, a_t)$  table
       for  $s_t$  and record as best action, best value. Update  $N(s_{t-1}, a_{t-1})$ , increment
       the number of visits by 1. Update the estimation of  $Q(s_{t-1}, a_{t-1})$  as:
       
$$Q(s_{t-1}, a_{t-1})_{new} \leftarrow Q(s_{t-1}, a_{t-1})_{old} + \beta [r_t + \alpha \cdot \text{best value} - Q(s_{t-1}, a_{t-1})_{old}]$$

9:     end while
10:  end while
11:  for All possible states do
12:    Check the updated  $Q(s, a)$  values from the table and determine the best ac-
       tions using the costs
13:    Update the learned policy and store for comparison
14:  end for
15:  Return the latest learned policy
16: end while

```

As the pseudo-code above summarizes, the Q-Learning algorithm works backward. In each episode, meaning in each simulation run, at the first decision epoch ($t = 1$), we determine the s_1 by checking the number of jobs in the system; we pick the action based on our strategy and calculate the reward. The action selection is based on the exploration strategy structure. For the Boltzman Strategy, based on the τ value and the existing estimated values in the $Q(s, a)$ table, we calculate probabilities of selecting our two actions using the Boltzmann function as explained in equation 3.21. Then, we pick the action based on the calculated probabilities by picking uniform random numbers between 0 and 1. For the ϵ -Greedy strategy given in equations 3.27 and 3.28, we distribute the probabilities between the better and the other actions. So, we set the probabilities of actions and select the action again by picking uniform random numbers. For the UCB exploration strategy, no randomness is involved. Using the hyperparameter c , we calculate a penalty based on the number of visits to the state divided by the number of action selections as in equation 3.29. It favors the action picked less by subtracting a higher value from it. After the values are obtained, the minimum one is picked as our action. The reward calculation is as in equation 3.25. In our implementation, we carefully aligned the time aspect of the cost calculation. After the time step is greater than or equal to two we now have the information of the next steps. In $t = 2$ we know s_{1+1} , in $t = 3$ we know s_{2+1} so from now on we can calculate the $Q(s_{t-1}, a_{t-1})$ estimates. Now for each decision epoch we repeat the process of recalling the s_{t-1} and a_{t-1} values, recording the s_t, a_t, r_t values to the relevant sets, and updating the estimate of $Q(s_{t-1}, a_{t-1})$ in the table. To do the update, we first calculate the $\min_a Q(s_t, a)$ by checking our previous estimates from the table and recording the value. Then we use it together with r_t and the old estimate values of $Q(s_{t-1}, a_{t-1})$ to obtain the new $Q(s_{t-1}, a_{t-1})$ estimation. We repeat this process until the simulation ends. After the simulation is finished, we determine the learned policy by using the $Q(s, a)$ estimates learned until now. Picking the action with minimum cost gives us the learned policy. We then store this policy as our learned policy and repeat the whole process with a new episode, meaning a new simulation run, until a termination condition is accomplished. A new simulation run or a new episode is called as one iteration of the algorithm. Since every iteration means a new simulation, and we keep our table

estimated throughout the iterations, we can think of every simulation as a new sample from all possible queueing trajectories. Thus, Q-Learning estimates the optimum expected average cost values or state-action values with this sampling strategy [19]. Also, as mentioned in detail before, Q-Learning makes this estimate with a technique that is very similar to the Value Iteration algorithm. The main difference is that we use simulation samples and sampled values instead of known probabilities to calculate it. Based on the similarities of the Q-Learning algorithm to value iteration, we

Algorithm 2 Q-Learning: Addition of the Second Order Difference Termination

```

1: for All possible states do
2:   Check the updated  $Q(s_t, a_t)$  values from the table and determine the best action
     using the costs
3:   Update the learned policy and store for comparison
4: end for
5: Using the learned policy and  $Q(s, a)$  table find the  $\tilde{V}_\pi^{n+1}$  values
6: Calculate the difference 1 with  $\tilde{V}_\pi^{n+1} - \tilde{V}_\pi^n$ 
7: Calculate the difference 2 with  $\tilde{V}_\pi^n - \tilde{V}_\pi^{n-1}$ 
8: Update the  $\tilde{V}_\pi^n$  as  $\tilde{V}_\pi^n = \tilde{V}_\pi^{n+1}$ 
9: Update the  $\tilde{V}_\pi^{n-1}$  as  $\tilde{V}_\pi^{n-1} = \tilde{V}_\pi^n$ 
10: Calculate the convergence criteria  $difference_2 - difference_1 = \theta$ 
11: if  $\theta > \sigma$  then
12:   Return the learned policy
13: else
14:   Continue to the beginning
15: end if

```

adapted the second-order differences. The second-order differences are calculated after the completion of each episode of the Q-Learning algorithm. We used the maximum of the second-order difference vector as our convergence value. The convergence value of 0.1 was used as the stopping point. Based on our experiments with different instances, after reaching the second-order difference of 0.1, the algorithm does not change its

results. However, one can use bigger or smaller values based on one's problems and cost values. Once the second order difference is smaller than the determined σ value (0.1), the algorithm terminates. The pseudo-code given as Algorithm 2 can be added to the end of the Q-Learning algorithm to add the Second Order Difference Termination functionality. In the code, $\tilde{V}_\pi^n$ represents the estimated action value that is obtained from the values on the $Q(s, a)$ table of the last selected actions by the algorithm. In other words, it is an estimate of the V_π^n values in the Value Iteration algorithm. After each episode by recording the values $\tilde{V}_\pi^n, \tilde{V}_\pi^{n-1}$ and $\tilde{V}_\pi^{n+1}$ we calculate the second order differences. And then if it is smaller than our threshold we terminate the algorithm.

3.3. Simulation

To create an environment in which we can train the Q-Learning algorithm and also to compare the resulting MDP and Q-Learning policies we developed an event-based simulation using the Simpy package of the Python computing environment. In this section we will explain, verify and validate our simulation model.

3.3.1. Simulation Verification and Validation

All variations of queue simulations was firstly validated manually by creating simulations of short durations making checks on the order of the events (arrivals and services). The main logic of the Simpy package includes an environment object in which it has its own methods to schedule events in the right time. This logic is based on the Python's Generator functions. Generator functions are kind of functions which can be iterated through, such as items of a list and also they have ability to freeze in time and then continue to work on later. Simpy's scheduler iterates through the function that generates arrivals. It generates arrivals by checking the clock and waiting a random inter-arrival time. It then goes on to iterate the next arrival and so on. At the same time every arrival generating function calls a function that handles the service of the arriving customer. The function that handles the service uses the Simpy's resource object which represents the server. It is used by all the customers commonly.

If the resource is occupied since the service function is also a generator function, it has the ability to stop and wait until the server is free. Thus the time a customer waits while the server is busy is recorded as the time she waits on queue. Simpy's scheduler schedules all these events in a first time first serve manner. All the logic behind the Simpy package and checking the system by hand rigorously validates the designed simulation structure. Figure 3.2 represents the simulation structure.



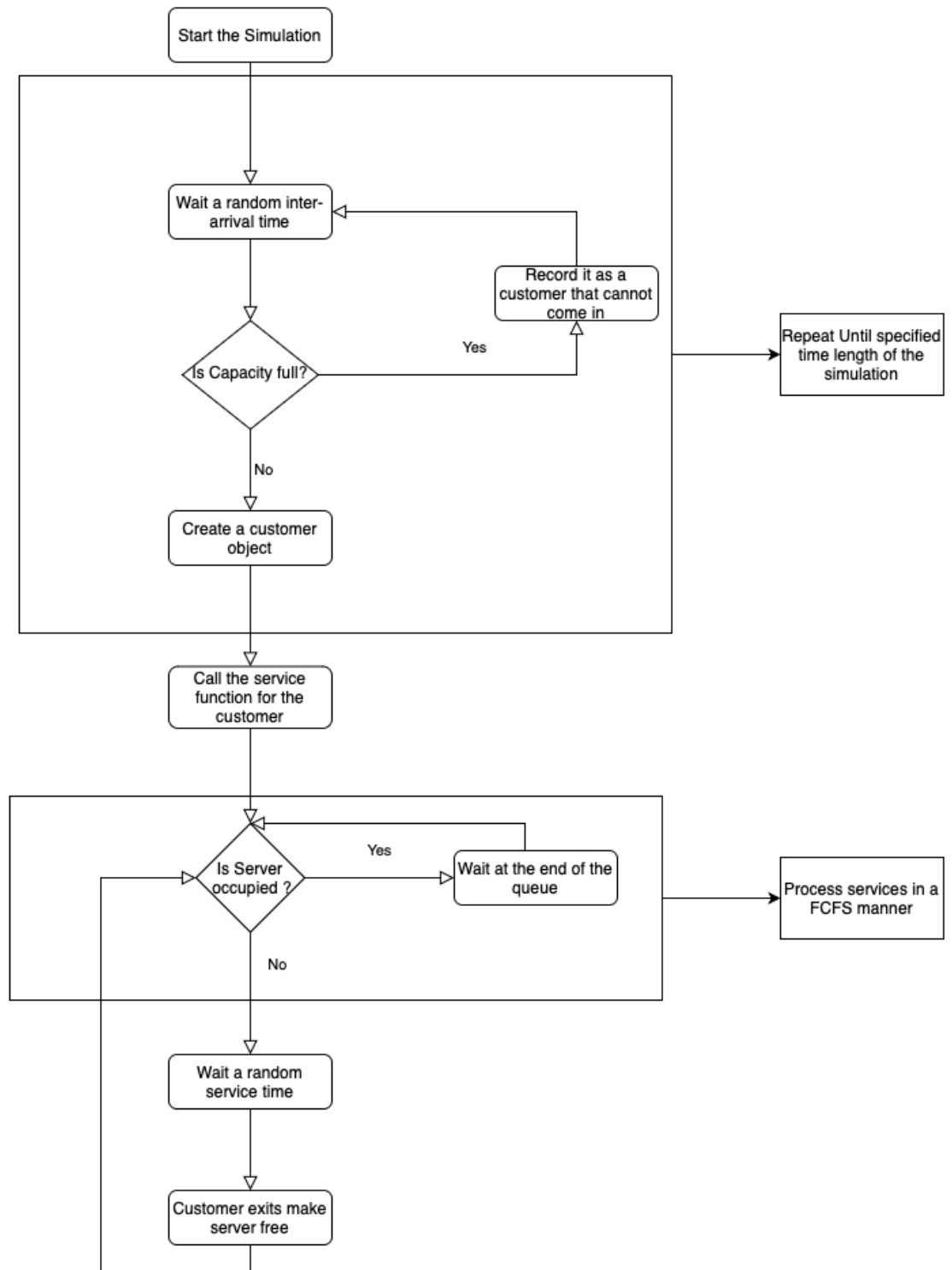


Figure 3.2. Simulation structure diagram

Python's Simpy package was used to create discrete event simulations of M/M/1/K and M/G/1/K models. In all of the variations, the same coding structure was used and only the distributions of the service times were changed. To verify that the code generates the desired queueing structure, first using a sample size of 50 runs, M/M/1 simulation data was collected and $L, L_q, \omega, \omega_q, \lambda, \mu$, and ρ were calculated. Then the results were compared with their corresponding theoretical values. Confidence intervals of 95% and 90% were used to verify that the theoretical calculations are within the range of obtained results.

Table 3.1. Performance Measures for $\rho = 0.9$ and 90% confidence.

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
L	9.0000	9.0117	0.9549	8.7853	9.2381
L_q	8.1000	8.1121	0.9491	7.8870	8.3371
ω	1.0000	1.0016	0.1020	0.9774	1.0258
ω_q	0.9000	0.9016	0.1017	0.8775	0.9257
λ	9.0000	8.9923	0.0486	8.9807	9.0038
μ	10.0000	9.9962	0.0505	9.9842	10.0082
ρ	0.9000	0.8996	0.0073	0.8979	0.9014

The simulation runs were performed with the length of 5000 time units which includes a 1000 time units of warm-up period. The test is repeated with different traffic intensities of $\rho = 0.25, 0.5, 0.75, 0.8$ and 0.9 . The obtained result for 90% confidence and $\rho = 0.9$ M/M/1 queue is represented in table 3.1. The result tables for the rest of the ρ values for M/M/1 queues are in Appendix section A. The results table verify that the M/M/1 simulation structure is correct and creates the desired data. Then to verify the M/M/1/K structure we designed a simulation with $\rho = 1$.

Table 3.2. M/M/1/K Probability values vs simulation values for $K = 10$ and $\rho = 1$.

K	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
0	0.090909091	0.090467	0.004955	0.089292	0.091642
1	0.090909091	0.090657	0.004338	0.089629	0.091686
2	0.090909091	0.090548	0.002727	0.089902	0.091195
3	0.090909091	0.090741	0.002741	0.090092	0.091391
4	0.090909091	0.090538	0.002471	0.089953	0.091124
5	0.090909091	0.090763	0.002209	0.090239	0.091287
6	0.090909091	0.090762	0.002063	0.090273	0.091252
7	0.090909091	0.090684	0.002284	0.090143	0.091226
8	0.090909091	0.091527	0.002962	0.090825	0.092230
9	0.090909091	0.091418	0.003493	0.090590	0.092246
10	0.090909091	0.091893	0.004287	0.090876	0.092909

Theoretically in that case the probability of visiting every state should be equal. Using a similar structure with 50 runs and run lengths of 10000-17500-37500 time units with 2000-3500-7500 time units as warm-up for $K = 10 - 15 - 25$ we collected the time the simulation spends in each state and created the probability distribution of states. Then we used a t-test to verify that the theoretical probabilities are within the 90% & 95% confidence interval of our calculated probabilities. This step verifies that our simulation code creates a valid M/M/1/K simulation. Table 3.2 shows the result for 90% confidence interval of an M/M/1/K with $K = 10$, the rest of the tables are in the appendix A. In the structure the nature of inter-arrival times and service times are created in Python using Numpy library. Using the same code we only changed the corresponding Numpy functions of random variates to develop a M/G/1/K simulation. Therefore verifying the M/M/1/K code with the explained t-tests also would verify M/G/1/K. Additionally the verification tables for $K = 10$ is as below the rest of the tables added to the Appendix section A.

4. Numerical Analysis

In this section, we will numerically compare the performance of the MDP model and RL models. We first compare the performances of MDP optimal model and Q-Learning models in the M/M/1/K instances to test the overall success of the Q-Learning models. Then, we use the best-performing Q-Learning model in the M/G/1/K instances to see its ability to solve harder problems. For the first part, in which we test the performance of Q-Learning in the M/M/1/K problem, we solve different problem instances both with MDP and 3 variations of the Q-Learning model. On the MDP model, we use the average rates of arrival and service as input while the Q-Learning models learn everything from scratch. For the comparison of performances in M/M/1/K setting, we use the policy evaluation technique, using the value iteration algorithm to calculate the average expected cost per decision value of the policies produced by the Q-Learning models. Using the value iteration algorithm, our undiscounted cost structure necessitates that for the algorithm to converge, we have to check the second-order differences, as mentioned in the model development section. The second-order differences approach zero as the number of iterations approaches infinity. This is because the first order differences approach to a constant value. The equation

$$d_1 = V(X)_{n+1} - V(X)_n \quad (4.1)$$

expresses the first difference, where the equation

$$d_2 = V(X)_n - V(X)_{n-1} \quad (4.2)$$

expresses the second difference then, as the iterations progresses the equation

$$\text{As } n \rightarrow \infty, d_1 \cong d_2 \rightarrow C \quad (4.3)$$

expresses the constant value that both of the differences approach. This constant value gives us the expected average cost per decision which is defined in equation 4.3 as C .

For the latter, we develop two different Q-Learning models that we call as MDP-informed model and the ordinary model. The MDP-informed model starts its training for M/G/1/K problem using the MDP optimal policy of the M/M/1/K problem. It takes the actions of the MDP policy as the exploiting action and has a changing probability of taking exploratory actions based on its hyperparameters and strategy. MDP-informed model uses this schema only in the first iteration and then continues as an ordinary Q-Learning model using its exploration strategy. The ordinary model is the same model we use in M/M/1/K instances. We do not apply a hyperparameter search for M/G/1/K cases both in the MDP-informed and ordinary model. We use the same hyperparameter values with the best-performing strategy of the M/M/1/K instances. We use simulations to compare performances in the M/G/1/K setting. To clarify beforehand, when we say iterations, for the Q-learning models we refer to the number of episodes or simulation runs that it takes for the model to learn. We record the produced policies, the number of iterations to converge, and used hyperparameter values. Then we use the simulation model to compare the performance of the policies produced by MDP and Q-Learning models. The equation

$$Avg.Cost.PerDecision = \frac{1}{T} \left[\sum_{t=1}^T s_t h + \mu_{t,i} c_i \right] \quad (4.4)$$

expresses the calculation of the costs in the M/G/1/K instances where t is the time s_t is the value of the state at time t which is the number of jobs, h is the holding cost per job, $\mu_{t,i}$ is the selected service rate $i = 1, 2$ at time (during the time t) and c_i is the cost of choosing the rate i . The cost is calculated in each decision epoch until the end of the simulation run. And then it is divided to simulation time to obtain the average cost per decision of the policy. As our performance metric, we use the average expected cost per decision which is defined in 4.4. In other words we can claim that we use the average expected cost per decision logic again but this time we use simulations to calculate an estimation of it for the M/G/1/K case.

To make sure that this metric which is calculated in simulations is valid and representative of the average cost per decision value calculated using the value iteration algorithm, we calculated the MDP policies' costs in simulation and observed that they approach the theoretical cost of the policies as the simulations become longer. We used the metric to compare our policies for M/G/1/K problems. The policies were compared in simulations and average cost per iteration values were noted. Using the simulation runs and the average cost per decision metric we compared the MDP policy, policy found by the MDP-informed Q-Learning model and the policy found by the ordinary Q-Learning model. In this comparison, to reduce the sample variance, based on the amount of data that is included in different lengths of the simulations different number of iterations were used to calculate the mean and standard deviations. Since higher ρ values mean more data per time unit, we changed the number of runs accordingly. As our metric to measure the amount of data we counted the number of decision epochs that existed in that simulation in average. Starting with 100 simulation runs for the problems with $\rho = 0.75$ for different lengths and ρ values we set the number of runs to provide nearly equal total data size which can be seen in table 4.1. Based on that we made simulations with the test lengths of 3125 time units with 625 (20%) of it being the warm-up period. This gives us a run length of 2500, then as seen on the table 4.1 we used the same lengths with changing number of runs to test obtain the confidence intervals on the subsequent chapters to compare the results.

Table 4.1. Data sizes for different simulation runs.

K	Number of Runs	Max ρ	Run Length	Data Per Episode	Total Data
10	100	0.75	2500	14955.06	1495506
10	100	0.75	5000	29923.34	2992334
10	100	0.75	7500	44821.81	4482181
15	100	0.75	2500	14999.21	1499921
15	100	0.75	5000	29995.30	2999530
15	100	0.75	7500	44992.24	4499224
25	100	0.75	2500	14985.21	1498521
25	100	0.75	5000	29662.06	2966206
25	100	0.75	7500	44978.58	4497858
10	35	0.90	2500	43525.60	1523396
10	35	0.90	5000	87132.96	3049653.6
10	35	0.90	7500	130756.81	4576488.35
15	35	0.90	2500	44307.80	1550773
15	35	0.90	5000	88857.60	3110016
15	35	0.90	7500	133151.56	4660304.6
25	35	0.90	2500	44897.36	1571407.6
25	35	0.90	5000	89911.52	3146903.2
25	35	0.90	7500	134868.76	4720406.6
10	25	0.95	2500	89383.76	2234594
10	25	0.95	5000	178660.60	4466515
10	25	0.95	7500	268053.32	6701333
15	25	0.95	2500	91821.16	2295529
15	25	0.95	7500	275875.26	6896881.5
15	25	0.95	5000	183935.32	4598383
25	25	0.95	2500	93796.32	2344908
25	25	0.95	5000	187636.08	4690902
25	25	0.95	7500	281407.92	7035198

4.1. Parameter Sets and Experimental Design

To study the efficiency and effectiveness of the Q-Learning algorithm, we created different instances of the problem. For the M/M/1/K instances we used $K = \{10, 15, 25\}$ ρ values of 0.75, 0.90 and 0.95. For $\rho = 0.75$ we used the arrival rates of $\lambda = 3$ and $\mu_1 = 4, \mu_2 = 5$, for $\rho = 0.90$ we used the arrival rates of $\lambda = 9$ and $\mu_1 = 10, \mu_2 = 11$ and finally for $\rho = 0.95$ we used the arrival rates of $\lambda = 19$ and $\mu_1 = 20, \mu_2 = 21$

Table 4.2. Experimental setup of the problems.

ρ	K	λ	$\mu_i rates$
0.75	10	3	4,5
0.75	15	3	4,5
0.75	25	3	4,5
0.90	10	9	10,11
0.90	15	9	10,11
0.90	25	9	10,11
0.95	10	10	19,20
0.95	15	10	19,20
0.95	25	10	19,20

Table 4.2 represents the used parameters in creating the problems. All three exploration strategies of Boltzman Exploration, ϵ -greedy exploration, and UCB(Upper Confidence Bound) exploration were tested with these instances. Starting with the Boltzman Strategy, we used τ values of 5-10-15-20 with respective decay rates of 0-0.1-0.05-0.01. We applied linear decay by decreasing the τ values after each iteration until a stopping value of 0.5. We selected these values to cover a wide range of possibilities based on our problem by experimenting with different values. We observed in our problem instances that using these τ values provides good starting points for exploration. For the ϵ -Greedy Strategy, the general approach on the literature [19] is to use smaller values such as 0.1, 0.05 and 0.01. To cover a broader range, we used starting ϵ values of 0.25-0.15-0.1-0.05 and decay rates of 0-0.001-0.0001-0.00001. We

used lower decay rates for this Strategy because the values in the literature are small, and the range is short. We applied the same linear decay approach using a stopping ϵ value of 0.01. Finally, for the UCB Strategy, we again used c values of 5-4-3-2 based on our experiments. Since our cost values were relatively smaller in our instances, using small c values was reasonable. We applied the same decaying Strategy to the c values using decay rates of 0-0.1-0.01-0.001 and a stopping c value of 0.5. Using 16 different hyperparameter values for each of the three strategies, we developed 18 different problem instances. We have created instances for three different K values, three different ρ values, and 2 different problems on which either the episode length changes or the costs change. In total, we tested 864 hyperparameter values for 18 different problems. Table 4.3 summarizes the used hyperparameter values of each Strategy. Table 4.3 summarizes the hyperparameter values used in our experiments.

Table 4.3. Hyperparameter settings for different strategies.

Strategy	Starting Values	Decay Rates	Stopping Values	Total Cases
Boltzman	[20,15,10,5]	[0,0.10,0.05,0.01]	0.1	288
ϵ -Greedy	[0.25,0.15,0.10,0.05]	[0,0.001,0.0001,0.00001]	0.01	288
UCB	[5,4,3,2]	[0,0.1,0.01,0.001]	0.5	288

The tested problems' service costs and holding costs parameters are set to be difficult enough by checking the optimal solution of the problem and the convergence of the value iteration algorithm. We experimented with different parameters to create hard enough problems which can be seen in table 4.4. As seen in the table we used a similar pattern to create our service costs in problems. We used the arrival rate's numerical value and doubled value as service costs in all problems. For instance, for the problems with the arrival rate of three, we set the cost of serving with rate 4 as 3 and with rate 5 as 6. We applied this pattern in all of our problems. Then, we picked suitable holding costs to create challenging problems.

To keep track of different versions of the problems, a name addressing system is generated. The numbering system consists of the following specifications regarding the problem with the following order : Arrival Rate / Capacity (K) / Maximum ρ / Holding Cost / Episode Length. Since the service costs can be calculated using the same pattern explained above, addressing the problems with these parameters is enough to understand the problem. Table 4.4 summarizes the experimented problems. To simplify the matter, we can calculate a cost for a defined time step for the experimented problem 3-10-0.75-15-2500. As we mentioned before, the cost calculation in the continuous system requires knowledge of the system state's and service duration's time duration. However, to simplify the matter, let us assume a discrete setting and calculate the cost. The cost or reward that is incurred per iteration can be calculated as $s_t h + \mu_{t,i} c_i$ if we assume a discrete setting in which t is the time step (decision epoch) s_t is the value of the state at time t which is the number of jobs, h is the holding cost per job, $\mu_{t,i}$ is the selected service rate $i = 1, 2$ at time t and c_i is the cost of choosing the rate i . e.g. given that there are 3 jobs in the system and the selected rate is 5 of the experimented problem 3-10-0.75-15-2500, the cost is 75. Table 4.4 summarizes the experimented problems and the numbering system.

Finally, for other hyperparameters our preferences is as follows. We experimented with different lengths for training episodes. We used the second order difference method as explained before for the termination criterion of the MDP model and Q-Learning algorithms. Since the MDP model uses the average rates while Q-Learning models work with its estimates, the convergence termination thresholds differ. We chose 10^{-7} for the value iteration algorithm, while for the Q-Learning model, we chose 10^{-1} based on our experiments with problems as explained in the methodology section. Since one of the study aims is to show the practical usage of this algorithm, the average time required per episode in different strategies and scenarios was also recorded. The best strategy and hyperparameter values for each scenario were addressed using the criteria of faster convergence, smaller time required per iteration, and the quality of the found policy. The quality of the found policy is based on the value of the policy calculated

Table 4.4. Summary of the experimented problems.

ρ	K	λ	μ_i rates	Holding Costs	Service Costs	Ep.Length	Experiments
0.75	10	3	[4,5]	15	[3,6]	2500	3-10-0.75-15-2500
0.75	10	3	[4,5]	5	[3,6]	2500	3-10-0.75-5-2500
0.75	15	3	[4,5]	3	[3,6]	2500	3-15-0.75-3-2500
0.75	15	3	[4,5]	10	[3,6]	2500	3-15-0.75-10-2500
0.75	25	3	[4,5]	1.8	[3,6]	2500	3-25-0.75-1.8-2500
0.75	25	3	[4,5]	1.8	[3,6]	2500	3-25-0.75-1.8-2500
0.90	10	9	[10,11]	75	[9,18]	2500	9-10-0.90-75-2500
0.90	10	9	[10,11]	45	[9,18]	2500	9-10-0.90-45-2500
0.90	15	9	[10,11]	30	[9,18]	2500	9-15-0.90-30-2500
0.90	15	9	[10,11]	30	[9,18]	5000	9-15-0.90-30-5000
0.90	25	9	[10,11]	35	[9,18]	2500	9-25-0.90-35-2500
0.90	25	9	[10,11]	35	[9,18]	7500	9-25-0.90-35-7500
0.95	10	19	[20,21]	375	[19,38]	2500	19-10-0.95-375-2500
0.95	10	19	[20,21]	375	[19,38]	5000	19-10-0.95-375-5000
0.95	15	19	[20,21]	185	[19,38]	2500	19-15-0.95-185-2500
0.95	15	19	[20,21]	185	[19,38]	5000	19-15-0.95-185-5000
0.95	25	19	[20,21]	70	[19,38]	2500	19-25-0.95-70-2500
0.95	25	19	[20,21]	70	[19,38]	5000	19-25-0.95-70-5000

by the value iteration algorithm for the M/M/1/K instances and the average cost per decision metric calculated using simulations for the M/G/1/K instances.

4.2. Results of M/M/1/K Experiments

4.2.1. Performance Comparison of the Q-Learning Models with MDP

In this section we will compare the overall performance of Q-Learning models compared to the optimal results of the MDP. As mentioned earlier the performance metric is the average expected cost per decision, which is calculated using the value iteration algorithm for each policy.

Starting with the performance of the Boltzman Strategy we see that in Table 4.5 in two instances 3.0-10-0.75-5-2500 and 9.0-25-0.9-35-7500 the model finds the optimal

policy. For the rest of the instances, we see that the costs of the policies found by the Boltzman Strategy are very close to the optimal costs. The worst result is for the instance 3.0-15-0.75-3-2500 in which the model deviates nearly 3 percent from the optimal result. The average percent deviation per instance is 0.668. We observe that as the ρ value of the instances increases the deviation seems to decrease which is reasonable since higher ρ values mean more data per training episode given the equal simulation length. We also observe this for the instances with $\rho = 0.75$ and 0.90 as the episode length increases the performance improves. However, surprisingly for the instances with $\rho = 0.95$ increasing the episode length decreases the model performance. Considering the number of iterations to converge and the total convergence time, we see that on average it takes 37.88 iterations which translates to 0.623 hours on average per instance for the model to converge. We see that as ρ increases both the number of iterations and the time increases. At most, it takes 5.411 hours for the model to converge which is for the instance 19.0-25-0.95-70-5000.

Table 4.5. Comparison of the results of the Boltzman Strategy and MDP.

Problem	Opt.Cost	Q-LearnCost	Difference	Difference %	IterToConv	TimeToConv(H)
3.0-10-0.75-15-2500	34,769	34,769	0.000	0.000	20	0.005
3.0-10-0.75-5-2500	17,883	18,220	0.337	1.882	14	0.003
3.0-15-0.75-10-2500	27,242	27,321	0.078	0.287	50	0.013
3.0-15-0.75-3-2500	14,012	14,409	0.397	2.834	35	0.009
3.0-25-0.75-1.8-2500	11,160	11,331	0.171	1.532	25	0.006
3.0-25-0.75-1.8-5000	11,160	11,169	0.010	0.087	39	0.030
9.0-10-0.9-45-2500	222,580	223,625	1.045	0.469	22	0.034
9.0-10-0.9-75-2500	324,528	326,259	1.731	0.533	25	0.039
9.0-15-0.9-30-2500	197,645	198,148	0.503	0.255	14	0.023
9.0-15-0.9-30-5000	197,645	197,782	0.136	0.069	58	0.094
9.0-25-0.9-35-2500	240,427	242,661	2.233	0.929	16	0.027
9.0-25-0.9-35-7500	240,427	240,427	0.000	0.000	63	1.452
19.0-10-0.95-375-2500	1856,759	1861,058	4.299	0.232	50	0.210
19.0-10-0.95-375-5000	1856,759	1866,239	9.480	0.511	44	1.144
19.0-15-0.95-185-2500	1357,740	1365,334	7.594	0.559	24	0.202
19.0-15-0.95-185-5000	1357,740	1366,271	8.531	0.628	54	2.230
19.0-25-0.95-70-2500	852,857	858,352	5.495	0.644	29	0.281
19.0-25-0.95-70-5000	852,857	857,772	4.915	0.576	100	5.411

Considering the performance of the ϵ -Greedy Strategy, we see in Table 4.6 that it cannot find the optimal solution in any of the cases. In general, we see that it is worse than the Boltzman Strategy in terms of costs. The worst result it obtained is for the instance 9.0-25-0.9-35-7500 in which the model deviates nearly 11 percent from the optimal result. The average percent deviation per instance is 2.594 which is higher than the Boltzman. We observe that as the ρ value of the instances increases the deviation seems to decrease similar to the case in Boltzman Strategy. However different from Boltzman, we observe that for the instances with $\rho = 0.75$ and 0.90 as the episode length increases, the performance decreases but, for the instances with $\rho = 0.95$ increasing the episode length improves the model performance. In terms of the number of iterations to converge and the convergence time, we see that on average it takes 42 iterations and 0.460 hours per instance for the model to converge. Again We see that as ρ increases both the number of iterations and the time increases. At most, it takes 2.259 hours for the model to converge which is for the instance 19.0-25-0.95-70-5000. It seems that though performed worse ϵ -Greedy seems to converge quicker than the Boltzman Strategy.

Table 4.6. Comparison of of the results of ϵ -Greedy Strategy and MDP.

Problem	Opt.Cost	Q-LearnCost	Difference	Difference %	IterToConv	TimeToConv(H)
3.0-10-0.75-15-2500	34.769	36.412	1.642	4.510	43	0.876
3.0-10-0.75-5-2500	17.883	18.187	0.304	1.671	17	0.876
3.0-15-0.75-10-2500	27.242	28.913	1.671	5.779	48	0.689
3.0-15-0.75-3-2500	14.012	14.254	0.241	1.694	29	0.689
3.0-25-0.75-1.8-2500	11.160	11.213	0.054	0.479	45	0.718
3.0-25-0.75-1.8-5000	11.160	11.278	0.118	1.045	29	1.775
9.0-10-0.9-45-2500	222.580	222.837	0.257	0.115	16	4.389
9.0-10-0.9-75-2500	325.000	332.351	7.351	2.212	22	4.389
9.0-15-0.9-30-2500	197.645	202.862	5.216	2.571	29	5.109
9.0-15-0.9-30-5000	197.645	203.154	5.509	2.712	26	24.469
9.0-25-0.9-35-2500	240.427	263.167	22.740	8.641	109	5.208
9.0-25-0.9-35-7500	240.427	270.451	30.024	11.101	53	68.830
19.0-10-0.95-375-2500	1856.759	1862.431	5.672	0.305	47	24.088
19.0-10-0.95-375-5000	1856.759	1859.435	2.677	0.144	44	168.846
19.0-15-0.95-185-2500	1357.740	1368.615	10.875	0.795	47	27.424
19.0-15-0.95-185-5000	1357.740	1368.548	10.808	0.790	28	171.391
19.0-25-0.95-70-2500	852.857	863.153	10.296	1.193	68	24.455
19.0-25-0.95-70-5000	852.857	860.920	8.064	0.937	50	162.648

Finally, for UCB Strategy we again cannot see an optimal result. In general, we see that in table 4.7 it is worse than the Boltzman Strategy. The worst result is for the instance 9.0-25-0.9-35-7500 in which the model deviates nearly 11 percent from the optimal result. The average percent deviation per instance is 2.352 which is higher than the Boltzman but lower than *epsilon*-Greedy Strategy. On the contrary to other two strategies it performed better on instances with $\rho = 0.75$ which is interesting. The worst results belong to instances with $\rho = 0.90$ and $\rho = 0.95$. In terms of the number of iterations to converge and the convergence time, we see that in average it takes 45,27 iterations and 0.417 hours per instance for the model to converge. Similar to other 2 We see that as ρ increases both the number of iterations and the time increases. At most it takes 1.853 hours for the model to converge which is for the instance 19.0-10-0.95-375-5000. It seems that UCB is the fastest to converge in terms of time among these three strategies.

Table 4.7. Comparison of the results of UCB Strategy and MDP.

Problem	Opt.Cost	Q-LearnCost	Difference	Difference %	IterToConv	TimeToConv(H)
3.0-10-0.75-15-2500	34,769	35,650	0.880	2.532	39	0.014
3.0-10-0.75-5-2500	17,883	18,219	0.336	1.880	18	0.006
3.0-15-0.75-10-2500	27,242	28,198	0.955	1.074	89	0.032
3.0-15-0.75-3-2500	14,012	14,267	0.255	1.820	37	0.013
3.0-25-0.75-1.8-2500	11,160	11,227	0.067	0.603	54	0.020
3.0-25-0.75-1.8-5000	11,160	11,294	0.134	1.204	24	0.018
9.0-10-0.9-45-2500	222,580	223,153	0.573	0.257	19	0.035
9.0-10-0.9-75-2500	325,000	329,263	4.263	1.312	29	0.053
9.0-15-0.9-30-2500	197,645	201,798	4.152	2.101	36	0.076
9.0-15-0.9-30-5000	197,645	202,683	5.038	2.549	30	0.196
9.0-25-0.9-35-2500	240,427	266,552	26.125	10.866	90	0.189
9.0-25-0.9-35-7500	240,427	267,648	27,221	11.322	71	1.391
19.0-10-0.95-375-2500	1,856,759	1,864,403	7,645	0.412	60	0.425
19.0-10-0.95-375-5000	1,856,759	1,866,084	9,326	0.502	46	1.853
19.0-15-0.95-185-2500	1,357,740	1,370,243	12,503	0.921	52	0.352
19.0-15-0.95-185-5000	1,357,740	1,366,227	8,487	0.625	39	1.736
19.0-25-0.95-70-2500	852,857	863,966	11,109	1.303	42	0.169
19.0-25-0.95-70-5000	852,857	861,894	9,037	1.060	40	0.923

Evaluating all the results together in general the winner seems to be the Boltzman Strategy when we consider the quality of the results and convergence times all together. Boltzman results are better in average than the other two, though it seems to be slowest to converge. The difference on the convergence times is not that big especially if we deem the one case 19.0-25-0.95-70-5000 that takes too much time to converge as an outlier. On the other hand ϵ -Greedy and UCB Strategies seem to have close results to each other in terms of solution quality. In average their results are close and also in deviations from optimal seem they have similar behavior. UCB strategy has some poor results in instances 9.0-25-0.9-35-2500 and 9.0-25-0.9-35-7500. Though the difference is not much from ϵ -Greedy Strategy, UCB is the fastest to converge. Considering that, we think that the second best performer is UCB Strategy and ϵ -Greedy Strategy is the third.

We continue with the examination of the policies that have been found by the Q-Learning models. The table 4.8 shows the best policies of the Boltzman model and the MDP policies. The policy vectors are column vectors however they were showed in the row format to use the space effectively. When we examine the policies found by respective methods in general MDP policies and Boltzman Strategy's policies are very similar. Generally the difference is only a few decisions. The general structure shows us that both MDP and the model changes the rates back and forth accordingly. The table 4.9 and 4.10 shows the best policies of ϵ -Greedy and UCB strategies respectively. Both for the ϵ -Greedy and UCB Strategies the same things are valid however we observe that they generally include more decisions that are different than the optimal.

Table 4.10. Comparison of MDP policies and the best policies of the UCB Strategy.

Problem	MDP Policy	UCB Best Policy
3.0-10-0.75-15-2500	[4, 4, 5, 5, 5, 5, 5, 5, 5, 5]	[4, 4, 4, 5, 5, 5, 5, 5, 5, 4]
3.0-10-0.75-5-2500	[4, 4, 4, 4, 5, 5, 5, 5, 5, 4]	[4, 4, 4, 4, 4, 5, 4, 4, 5, 5, 4]
3.0-15-0.75-10-2500	[4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]	[4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 4, 4, 5, 4, 5]
3.0-15-0.75-3-2500	[4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 4]	[4, 4, 4, 4, 4, 4, 4, 5, 5, 4, 4, 4, 4, 4, 4]
3.0-25-0.75-1.8-2500	[4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4]	[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 4, 5, 4, 4, 4, 4, 4]
3.0-25-0.75-1.8-5000	[4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4]	[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 4, 4, 4, 5, 4, 4, 4, 5, 5]
9.0-10-0.9-45-2500	[10, 10, 10, 10, 10, 11, 11, 11, 10, 10, 10]	[10, 10, 10, 10, 10, 10, 11, 10, 10, 10, 10]
9.0-10-0.9-75-2500	[10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 10]	[10, 10, 11, 11, 11, 11, 10, 11, 11, 10, 10]
9.0-15-0.9-30-2500	[10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 10, 10]	[10, 10, 10, 10, 11, 10, 10, 11, 10, 11, 10, 11, 11, 10, 10, 10]
9.0-15-0.9-30-5000	[10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 10, 10]	[10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 10, 10, 10, 11, 10, 10]
9.0-25-0.9-35-2500	[10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11]	[10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 10, 11, 11, 11, 10, 11, 10, 10, 11, 11, 11, 11, 10, 10]
9.0-25-0.9-35-7500	[10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11]	[10, 10, 10, 10, 10, 10, 10, 11, 11, 10, 11, 11, 10, 10, 10, 10, 11, 10, 10, 11, 11, 11, 10, 10, 10]
19.0-10-0.95-375-2500	[20, 20, 20, 20, 21, 21, 21, 21, 21, 20, 20]	[20, 20, 20, 20, 21, 21, 20, 21, 21, 21, 20]
19.0-10-0.95-375-5000	[20, 20, 20, 20, 21, 21, 21, 21, 21, 20, 20]	[20, 20, 20, 21, 21, 20, 21, 20, 21, 20, 20, 20]
19.0-15-0.95-185-2500	[20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 20, 20, 20]	[20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 20, 20, 20, 21, 20]
19.0-15-0.95-185-5000	[20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 20, 20, 20]	[20, 20, 20, 20, 20, 21, 21, 21, 21, 20, 20, 20, 21, 20, 20, 20, 20]
19.0-25-0.95-70-2500	[20, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 20, 20, 20, 20]	[20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 20, 20, 20, 20, 20, 20, 21, 21, 20, 20, 20, 20]
19.0-25-0.95-70-5000	[20, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 20, 20, 20, 20]	[20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 20, 20, 21, 20, 21, 20, 20, 20, 20, 20, 20, 20]

Though in average Boltzman Strategy seem to provide the best results fro Q-Learning, we will continue with a more detailed analysis and compare three strategies case by case in the next section to pick the best strategy in a more precise manner.

4.2.2. Performance Evaluations of Q-Learning Models

In the previous section we compared Q-Learning models with the MDP model and saw that Boltzman performed better in general comparing with the other two. In this section our aim is to find the best strategy among our 3 models together with the best hyperparameter values by examining the instances in a more detailed approach. By grouping the problems based on the ρ values we compare all models based on their performances using their costs and the percent deviation from the optimal MDP cost.

Table 4.11. Comparison of exploration strategies for the instances with $\rho = 0.75$.

Problem	Strategy	Q-LearnCost	Difference %	Iter. To Conv.	Time-To Conv. (H)
3.0-10-0.75-15-2500	Boltzman	34.769	0.000	20	0.005
3.0-10-0.75-15-2500	UCB	35.650	2.532	39	0.014
3.0-10-0.75-15-2500	E-Greedy	36.412	4.510	43	0.010
3.0-10-0.75-5-2500	E-Greedy	18.187	1.671	17	0.004
3.0-10-0.75-5-2500	UCB	18.219	1.880	18	0.006
3.0-10-0.75-5-2500	Boltzman	18.220	1.882	14	0.003
3.0-15-0.75-10-2500	Boltzman	27.321	0.287	50	0.013
3.0-15-0.75-10-2500	UCB	28.198	1.074	89	0.032
3.0-15-0.75-10-2500	E-Greedy	28.913	5.779	48	0.009
3.0-15-0.75-3-2500	E-Greedy	14.254	1.694	29	0.006
3.0-15-0.75-3-2500	UCB	14.267	1.820	37	0.013
3.0-15-0.75-3-2500	Boltzman	14.409	2.834	35	0.009
3.0-25-0.75-1.8-2500	E-Greedy	11.213	0.479	45	0.009
3.0-25-0.75-1.8-2500	UCB	11.227	0.603	54	0.020
3.0-25-0.75-1.8-2500	Boltzman	11.331	1.532	25	0.006
3.0-25-0.75-1.8-5000	Boltzman	11.169	0.087	39	0.030
3.0-25-0.75-1.8-5000	E-Greedy	11.278	1.045	29	0.014
3.0-25-0.75-1.8-5000	UCB	11.294	1.204	24	0.018

Starting on the problem instances with $\rho = 0.75$ we see in table 4.11 that among the 6 instances Boltzman and ϵ -Greedy Strategies were the closest to MDP results for 3 different instances. Considering the time it takes to converge all strategies seem to be very close to each other, we see that again Boltzman Strategy was the fastest in 3 of the instances while ϵ -Greedy Strategy was the fastest for the other 3.

Continuing with the problem instances with $\rho = 0.90$ we see in table 4.12 that among the 6 different cases the Boltzman Strategy has the best solution. Only for the instance 9.0-10-0.9-45-2500 ϵ -Greedy is better. In terms of the convergence times we see that there is not much difference between the instances. In 3 of the instances Boltzman Strategy and in the other 3 of the instances ϵ -Greedy is the fastest.

Table 4.12. Comparison of exploration strategies for the instances with $\rho = 0.9$.

Problem	Strategy	Q-LearnCost	Difference %	Iter. To Conv.	Time-To Conv. (H)
9.0-10-0.9-45-2500	E-Greedy	222.837	0.115	16	0.020
9.0-10-0.9-45-2500	UCB	223.153	0.257	19	0.035
9.0-10-0.9-45-2500	Boltzman	223.625	0.469	22	0.034
9.0-10-0.9-75-2500	Boltzman	326.259	0.533	25	0.039
9.0-10-0.9-75-2500	UCB	329.263	1.312	29	0.053
9.0-10-0.9-75-2500	E-Greedy	332.351	2.212	22	0.027
9.0-15-0.9-30-2500	Boltzman	198.148	0.255	14	0.023
9.0-15-0.9-30-2500	UCB	201.798	2.101	36	0.076
9.0-15-0.9-30-2500	E-Greedy	202.862	2.571	29	0.041
9.0-15-0.9-30-5000	Boltzman	197.782	0.069	58	0.094
9.0-15-0.9-30-5000	UCB	202.683	2.549	30	0.196
9.0-15-0.9-30-5000	E-Greedy	203.154	2.712	26	0.177
9.0-25-0.9-35-2500	Boltzman	242.661	0.929	16	0.027
9.0-25-0.9-35-2500	E-Greedy	263.167	8.641	109	0.158
9.0-25-0.9-35-2500	UCB	266.552	10.866	90	0.189
9.0-25-0.9-35-7500	Boltzman	240.427	0.000	63	1.452
9.0-25-0.9-35-7500	E-Greedy	270.451	11.101	53	1.013
9.0-25-0.9-35-7500	UCB	267.648	11.322	71	1.391

Finally for the instances with the $\rho = 0.95$ we see in table 4.13 that in 4 of the 6 instances Boltzman Strategy was the best and for the other 2 instances ϵ -Greedy Strategy was the best for the instance 19.0-10-0.95-375-5000 and UCB Strategy was the best for the instance 19.0-15-0.95-185-5000. In terms of the convergence times Boltzman Strategy was the fastest in 3 instances while UCB Strategy was the fastest for 2 instances and ϵ -Greedy Strategy was the fastest for 1 instance.

In the end we see that in 12 of the 18 instances Boltzman Strategy was the best Strategy while ϵ -Greedy Strategy was the best strategy in 5 instancens. The UCB strategy was the best strategy only for one instance. After examining the results in more detail we can see clearly that the Boltzman Strategy was the best strategy among these three strategies. Both in average results and in one by one comparisons Boltzman seem to works better.

Table 4.13. Comparison of exploration strategies for the instances with $\rho = 0.95$.

Problem	Strategy	Q-LearnCost	Difference %	Iter. To Conv.	Time-To Conv. (H)
19.0-10-0.95-375-2500	Boltzman	1861.0577	4.299	50	0.210
19.0-10-0.95-375-2500	ϵ -Greedy	1862.4309	5.672	47	0.314
19.0-10-0.95-375-2500	UCB	1864.4033	7.645	60	0.425
19.0-10-0.95-375-5000	ϵ -Greedy	1859.4351	2.677	44	2.064
19.0-10-0.95-375-5000	UCB	1866.0842	9.326	46	1.853
19.0-10-0.95-375-5000	Boltzman	1866.2389	9.480	44	1.144
19.0-15-0.95-185-2500	Boltzman	1365.3344	7.594	24	0.202
19.0-15-0.95-185-2500	ϵ -Greedy	1368.6149	10.875	47	0.358
19.0-15-0.95-185-2500	UCB	1370.2432	12.503	52	0.352
19.0-15-0.95-185-5000	UCB	1366.2266	8.487	39	1.736
19.0-15-0.95-185-5000	Boltzman	1366.2707	8.531	54	2.230
19.0-15-0.95-185-5000	ϵ -Greedy	1368.5481	10.808	28	1.333
19.0-25-0.95-70-2500	Boltzman	858.3520	5.495	29	0.281
19.0-25-0.95-70-2500	ϵ -Greedy	863.1527	10.296	68	0.462
19.0-25-0.95-70-2500	UCB	863.9661	11.109	42	0.169
19.0-25-0.95-70-5000	Boltzman	857.7717	4.915	100	5.411
19.0-25-0.95-70-5000	ϵ -Greedy	860.9205	8.064	50	2.259
19.0-25-0.95-70-5000	UCB	861.8942	9.037	40	0.923

After this overall evaluation we want to examine the average time it takes for one iteration for all 3 strategies. The average time per iteration in different scenarios was recorded in the tables 4.14,4.15 and 4.16 for the different ρ values of 0.75,0.90 and 0.95 in order. The experiments made on a personal computer. The test computer was a Macbook Air (2020) with M1 chip that has 8 cores and 16 gigabytes of Ram. The recordings was made with 100 iterations for the $\rho = 0.75$, 35 iterations for $\rho = 0.90$ and 25 iterations for $\rho = 0.95$. Here the iteration means 1 iteration of the Q-Learning Model which is one simulation run. The number of runs were selected like that as explained in the beginning of this section in table 4.1.

We observe the possible instances with $\rho = 0.75$ in Table 4.14. In 8 of the 9 instances ϵ -Greedy Strategy has the lowest time per iteration, only in for the first case Boltzman Strategy beats that.

Table 4.14. Average time per episode of strategies for the instances with $\rho = 0.75$.

K	Avg.TimePerEpisode(sec.)	Strategy	Avg.Dec.Ep.Num.	Episode Length
10	0.817	Boltzman	14955.06	2500
10	0.876	ϵ -Greedy	14955.06	2500
10	1.288	UCB	14955.06	2500
10	1.781	ϵ -Greedy	29923.34	5000
10	2.292	Boltzman	29923.34	5000
10	2.780	UCB	29923.34	5000
10	3.665	ϵ -Greedy	44821.81	7500
10	4.617	UCB	44821.81	7500
10	4.657	Boltzman	44821.81	7500
15	0.689	ϵ -Greedy	14999.21	2500
15	0.931	Boltzman	14999.21	2500
15	1.286	UCB	14999.21	2500
15	1.674	ϵ -Greedy	29995.30	5000
15	2.375	Boltzman	29995.30	5000
15	2.684	UCB	29995.30	5000
15	3.520	ϵ -Greedy	44992.24	7500
15	4.508	UCB	44992.24	7500
15	5.015	Boltzman	44992.24	7500
25	0.718	ϵ -Greedy	14985.21	2500
25	0.904	Boltzman	14985.21	2500
25	1.325	UCB	14985.21	2500
25	1.775	ϵ -Greedy	29662.06	5000
25	2.713	UCB	29662.06	5000
25	2.804	Boltzman	29662.06	5000
25	3.226	ϵ -Greedy	44978.58	7500
25	4.334	UCB	44978.58	7500
25	5.981	Boltzman	44978.58	7500

For the instances with $\rho = 0.90$, we observe in Table 4.15 in all of the cases Boltzman Strategy has the lowest time per iteration.

Table 4.15. Average Time Per Iteration of Strategies for the instances with $\rho = 0.90$

K	Avg.TimePerEpisode(sec.)	Strategy	Avg.Dec.Ep.Num.	Episode Length
10	1.942	Boltzman	43525.60	2500
10	4.389	ϵ -Greedy	43525.60	2500
10	6.615	UCB	43525.60	2500
10	8.891	Boltzman	87132.96	5000
10	17.316	ϵ -Greedy	87132.96	5000
10	22.117	UCB	87132.96	5000
10	27.218	Boltzman	130756.81	7500
10	65.073	ϵ -Greedy	130756.81	7500
10	65.817	UCB	130756.81	7500
15	2.040	Boltzman	44307.80	2500
15	5.109	ϵ -Greedy	44307.80	2500
15	7.580	UCB	44307.80	2500
15	8.823	Boltzman	88857.60	5000
15	23.490	UCB	88857.60	5000
15	24.469	ϵ -Greedy	88857.60	5000
15	29.216	Boltzman	133151.56	7500
15	71.590	UCB	133151.56	7500
15	73.180	ϵ -Greedy	133151.56	7500
25	2.154	Boltzman	44897.36	2500
25	5.208	ϵ -Greedy	44897.36	2500
25	7.552	UCB	44897.36	2500
25	9.502	Boltzman	89911.52	5000
25	21.942	ϵ -Greedy	89911.52	5000
25	27.205	UCB	89911.52	5000
25	29.034	Boltzman	134868.76	7500
25	68.830	ϵ -Greedy	134868.76	7500
25	70.535	UCB	134868.76	7500

For the instances $\rho = 0.95$ in Table 4.16 we again see in all of the cases Boltzman Strategy has the lowest time per iteration.

Table 4.16. Average Time Per Iteration of Strategies for the instances with $\rho = 0.95$

K	Avg.TimePerEpisode(sec.)	Strategy	Avg.Dec.Ep.Num.	Episode Length
10	3.777	Boltzman	89383.76	2500
10	24.088	ϵ -Greedy	89383.76	2500
10	25.523	UCB	89383.76	2500
10	23.393	Boltzman	178660.60	5000
10	145.035	UCB	178660.60	5000
10	168.846	ϵ -Greedy	178660.60	5000
10	59.779	Boltzman	268053.32	7500
10	368.885	UCB	268053.32	7500
10	391.820	ϵ -Greedy	268053.32	7500
15	7.591	Boltzman	91821.16	2500
15	24.382	UCB	91821.16	2500
15	27.424	ϵ -Greedy	91821.16	2500
15	37.163	Boltzman	183935.32	5000
15	160.252	UCB	183935.32	5000
15	171.391	ϵ -Greedy	183935.32	5000
15	81.812	Boltzman	275875.26	7500
15	376.983	UCB	275875.26	7500
15	464.704	ϵ -Greedy	275875.26	7500
25	8.713	Boltzman	93796.32	2500
25	14.523	UCB	93796.32	2500
25	24.455	ϵ -Greedy	93796.32	2500
25	48.697	Boltzman	187636.08	5000
25	83.038	UCB	187636.08	5000
25	162.648	ϵ -Greedy	187636.08	5000
25	48.697	Boltzman	281407.92	7500
25	189.433	UCB	281407.92	7500
25	319.913	ϵ -Greedy	281407.92	7500

Overall we can say that Boltzman Strategy has lower time per iteration values in general, for the instances with $\rho = 0.75$ ϵ -Greedy Strategy has lower values. As the ρ value increases ϵ -Greedy Strategy's time per iteration values increases drastically. We observe that tendency as, for $\rho = 0.75$ it has the lowest values in general, for $\rho = 0.90$ it has the second lowest values in most of the instances and finally for $\rho = 0.95$ it has the third lowest values.

We can say that all three of the strategies performed well overall. They produced policies that has near optimal performance results. However, after the comparisons in overall and in detail both with MDP optimal results and one another, we can say that

Boltzman Strategy was the best strategy in M/M/1/K problems. The second best is the ϵ -Greedy Strategy and the third best is the UCB Strategy. Now we will continue with the reporting of the best performing hyperparameter values of the each strategy.

4.2.3. Performance of the Boltzman Strategy

Here we report the best performed models for each strategy together with hyperparameter values. In the evaluation of the performance, our first criterion was the deviation from the optimal cost and the second one was the number of iterations to converge. In the case of a tie in average cost per iteration (which means that both models converged to the same policy) the model that required less iterations to converge was picked.

Table 4.17 lists the best models of the Boltzman Strategy. As discussed in earlier parts of this section the Boltzman Strategy is the most successful strategy. The hyperparameter value τ of the Boltzman Strategy values selected by experimenting different values. Since in this strategy Boltzman function directly works with the estimated values, the τ values are problem dependent. In our experiments we tried to experiment with a wide range of values. We observe in some cases such as 3.0-25-0.75-1.8-2500 and 3.0-25-0.75-1.8-5000 increase in episode length decreases the τ value of the best model. However there exist also other cases that τ stays the same or even decrease. Thus we cannot determine a pattern or relation here that says increasing the episode length decrease the necessity for more exploration.

Table 4.17. Boltzmann Strategy best performing models.

Problem	Strategy Best Parameters	Optimal Cost	Difference %	Iter. to Conv.
3.0-10-0.75-15-2500	Tau 5 - Decay - 0.1 - Stop 0.1	34,769	0,000	20
3.0-10-0.75-5-2500	Tau 5 - Decay - 0.1 - Stop 0.1	17,883	1,882	14
3.0-15-0.75-10-2500	Tau 20 - Decay - 0.01 - Stop 0.1	27,242	0,287	50
3.0-15-0.75-3-2500	Tau 5 - Decay - 0.05 - Stop 0.1	14,012	2,834	35
3.0-25-0.75-1,8-2500	Tau 10 - Decay - 0.1 - Stop 0.1	11,160	1,532	25
3.0-25-0.75-1,8-5000	Tau 5 - Decay - 0.1 - Stop 0.1	11,160	0,087	39
9.0-10-0.9-45-2500	Tau 20 - Decay - 0 - Stop 20	222,580	0,469	22
9.0-10-0.9-75-2500	Tau 10 - Decay - 0.05 - Stop 0.1	324,528	0,533	25
9.0-15-0.9-30-2500	Tau 5 - Decay - 0.05 - Stop 0.1	197,645	0,255	14
9.0-15-0.9-30-5000	Tau 5 - Decay - 0.1 - Stop 0.1	197,645	0,069	58
9.0-25-0.9-35-2500	Tau 5 - Decay - 0.05 - Stop 0.1	240,427	0,929	16
9.0-25-0.9-35-7500	Tau 5 - Decay - 0.1 - Stop 0.1	240,427	0,000	63
19.0-10-0.95-375-2500	Tau 15 - Decay - 0 - Stop 15	1.856,759	0,232	50
19.0-10-0.95-375-5000	Tau 20 - Decay - 0 - Stop 20	1.856,759	0,511	44
19.0-15-0.95-185-2500	Tau 15 - Decay - 0.1 - Stop 0.1	1.357,740	0,559	24
19.0-15-0.95-185-5000	Tau 20 - Decay - 0 - Stop 20	1.357,740	0,628	54
19.0-25-0.95-70-2500	Tau 15 - Decay - 0.1 - Stop 0.1	852,857	0,644	29
19.0-25-0.95-70-5000	Tau 10 - Decay - 0 - Stop 10	852,857	0,576	100

4.2.4. Performance of the ϵ -Greedy Strategy

Table 4.18 lists the best models of the ϵ -Greedy Strategy. As discussed in earlier parts of this section the ϵ -Greedy Strategy is the second best strategy. Again we observe that as the ρ values increase the number of iterations to converge seem to increase in average. The general approach on the literature advises the usage of smaller ϵ values such as 0.05 and 0.10. However in our models we observe that in 4 of our instances the best models used 0.25 and 0.15 as starting ϵ values. We also observe in some cases such as 19.0-10-0.95-375-2500 and 19.0-10-0.95-375-5000 increase in episode length decreases the ϵ value of the best model. However there exist also other cases that ϵ stays the same or even decrease. Again we cannot determine a pattern or relation here that says increasing the episode length decrease the necessity for more exploration.

Table 4.18. ϵ -Greedy Strategy best performing models.

Problem	Strategy Best Parameters	Optimal Cost	Difference %	Iter. to Conv.
3.0-10-0.75-15-2500	E-0.25-Dec-0.001-Stop-0.01	34.769	4.510	43
3.0-10-0.75-5-2500	E-0.10-Dec-0-Stop-0.1	17.883	1.671	17
3.0-15-0.75-10-2500	E-0.05-Dec-0-Stop-0.05	27.242	5.779	48
3.0-15-0.75-3-2500	E-0.05-Dec-0-Stop-0.05	14.012	1.694	29
3.0-25-0.75-1,8-2500	E-0.10-Dec-0.0001-Stop-0.01	11.160	0.479	45
3.0-25-0.75-1,8-5000	E-0.10-Dec-0.0001-Stop-0.01	11.160	1.045	29
9.0-10-0.9-45-2500	E-0.15-Dec-0.00001-Stop-0.01	222.580	0.115	16
9.0-10-0.9-75-2500	E-0.05-Dec-0.0001-Stop-0.01	325.000	2.212	22
9.0-15-0.9-30-2500	E-0.25-Dec-0.001-Stop-0.01	197.645	2.571	29
9.0-15-0.9-30-5000	E-0.05-Dec-0.001-Stop-0.01	197.645	2.712	26
9.0-25-0.9-35-2500	E-0.10-Dec-0-Stop-0.1	240.427	8.641	109
9.0-25-0.9-35-7500	E-0.10-Dec-0-Stop-0.1	240.427	11.101	53
19.0-10-0.95-375-2500	E-0.25-Dec-0.001-Stop-0.01	1856.759	0.305	47
19.0-10-0.95-375-5000	E-0.05-Dec-0-Stop-0.05	1856.759	0.144	44
19.0-15-0.95-185-2500	E-0.05-Dec-0.00001-Stop-0.01	1357.740	0.795	47
19.0-15-0.95-185-5000	E-0.10-Dec-0.0001-Stop-0.01	1357.740	0.790	28
19.0-25-0.95-70-2500	E-0.10-Dec-0.0001-Stop-0.01	852.857	1.193	68
19.0-25-0.95-70-5000	E-0.05-Dec-0.00001-Stop-0.01	852.857	0.937	50

4.2.5. Performance of Upper Confidence Bound (UCB) Strategy

We list the best models of the RL model that explores with UCB strategy, in table 4.19. As discussed in earlier parts of this section UCB Strategy was the worst performer among the three strategies. Again we observe that as the ρ values increase the number of iterations to converge seem to increase in average. Similar to the Boltzman Strategy, UCB applies a mathematical formula which is directly related with the estimated values of Q-Learning model. Therefore we picked our hyperparameter values by experimenting since they are problem dependent. Similar to other strategies we also see no correlation between the episode lengths and the degree of exploration which is controlled by the hyperparameter c .

Table 4.19. UCB Strategy Best Performing Models

Problem	Strategy Best Parameters	Optimal Cost	Difference %	Iter. To Conv.
3.0-10-0.75-15-2500	C-2-Dec-0.01-Stop-0.5	34,769	2,532	39
3.0-10-0.75-5-2500	C-5-Dec-0.001-Stop-0.5	17,883	1,880	18
3.0-15-0.75-10-2500	C-4-Dec-0.01-Stop-0.5	27,242	1,074	89
3.0-15-0.75-3-2500	C-4-Dec-0.01-Stop-0.5	14,012	1,820	37
3.0-25-0.75-1,8-2500	C-2-Dec-0-Stop-2	11,160	0,603	54
3.0-25-0.75-1,8-5000	C-5-Dec-0-Stop-5	11,160	1,204	24
9.0-10-0.9-45-2500	C-2-Dec-0.1-Stop-0.5	222,580	0,257	19
9.0-10-0.9-75-2500	C-2-Dec-0.01-Stop-0.5	325,000	1,312	29
9.0-15-0.9-30-2500	C-3-Dec-0.1-Stop-0.5	197,645	2,101	36
9.0-15-0.9-30-5000	C-4-Dec-0.1-Stop-0.5	197,645	2,549	30
9.0-25-0.9-35-2500	C-3-Dec-0.01-Stop-0.5	240,427	10,866	90
9.0-25-0.9-35-7500	C-2-Dec-0.001-Stop-0.5	240,427	11,322	71
19.0-10-0.95-375-2500	C-5-Dec-0.1-Stop-5	1,856,759	0,412	60
19.0-10-0.95-375-5000	C-2-Dec-0-Stop-2	1,856,759	0,502	46
19.0-15-0.95-185-2500	C-4-Dec-0.01-Stop-0.5	1,357,740	0,921	52
19.0-15-0.95-185-5000	C-4-Dec-0.1-Stop-0.5	1,357,740	0,625	39
19.0-25-0.95-70-2500	C-4-Dec-0.1-Stop-0.5	852,857	1,303	42
19.0-25-0.95-70-5000	C-3-Dec-0.1-Stop-0.5	852,857	1,060	40

4.3. Performance Results of M/G/1/K Instances

For the M/G/1/K extension of the problem, first, we have studied whether using the optimal policy of the M/M/1/K case as a starting solution increases the performance of the RL method in M/G/1/K problem. The objective was to test if hard-coding information based on the model knowledge improve the performance of a model-free RL algorithm. The second objective was to see whether the RL method performs good in a non-Markovian version of the problem as well without any hyperparameter tuning.

In this section, we will share the results of M/G/1/K extension of our model using the performance measure of average cost per decision defined earlier. We use the Boltzman Strategy in both of our models since it was the best-performing strategy as explained above. Our first model is the MDP-informed model, (MDPI). For this model, we use the MDP model's optimal policy obtained for M/M/1/K problem as the starting

policy. To use a starting policy mechanism which is not existent in Q-Learning models, the MDP-informed model uses the Boltzman Strategy differently in the first iteration. Normally, Boltzman Strategy uses Q-Learning estimation table and makes its action choice based on the estimated values. In MDPI, instead of deciding between the actions based on their estimated value, it decides between the MDP policy action and the other actions. The Boltzman mechanism of choosing the actions based on their values and creating a Boltzman distribution stays the same. Yet, we manually manipulate the decision values using a factor of 0.95 to make the action that has been prescribed by the MDP policy action to be picked more often. This factor was decided based on our experiments with our range of τ values. Using this factor we made sure that MDP policy's action will be selected with a probability of at least 70% to 90% based on the starting τ value. This modification was only for the first iteration so that model started with a biased learning pattern. This biased pattern makes the MDPI model start learning more about the MDP policy's actions. For the ordinary Q-Learning model, we used the ordinary Boltzman Strategy without any modification. In both of the models, hyperparameter values of the best-performing model's of the M/M/1/K trials were used. We did not make any new hyperparameter searches. We have tried 3 different service distributions. These distributions are deterministic, Erlang with the shape parameter $k = 2$ and the lognormal distribution. The distributions were picked to compare the performance of the Q-Learning models on distributions with different squared coefficients of variations. In increasing order, the squared coefficient of variation values are 0 for deterministic service, 0.5 for the erlang-2 distribution (1 for the exponential distribution M/M/1/K case), and 2 for the lognormal distribution, once we set the standard deviation value correctly. The mentioned models allow us to test the performance of an MDP informed model and also the ordinary model. Additionally, since we already presented the Q-Learning algorithm's performance for the M/M/1/K case in detail, as an extension showing the behavior of the model in non-Markovian settings using different service distributions with various squared-coefficient of variation values, is helpful to show its adaptability. A remark that should be noted again is that there is not a detailed hyperparameter search was done for the extension part, instead, we used the M/M/1/K parameters directly. As we will observe, this results in bad

performance on some of the instances as the model is sensitive to those parameters. Yet this disadvantage also tests the model's adaptability in practice. Assuming an already converged model in real life, giving a sudden change in the environment such as service distribution, this experiment may show us how the model continues to perform without changing anything in the model. The results change for the instances with different ρ values, similar to the M/M/1/K case. We will examine them in ascending order of ρ values.

We start our performance comparisons on the M/G/1/K instances by comparing the results of the Q-Learning Models which are MDP-Informed model and the ordinary model with the MDP optimal policy of the M/M/1/K model. We have used MDP optimal policy of M/M/1/K problem on our simulations and calculated the policy's cost in non-Markovian settings. Then we used this cost as a comparison metric both for MDP-Informed model and the ordinary model. Our motivation to calculate and use the cost of the MDP policy in non-Markovian settings is to have a benchmark. Using this benchmark we had the ability to test whether the MDP-Informed model improve its starting point or not, whether the ordinary model find a policy that outperforms MDP policy in a non-Markovian environment and finally we had the chance to compare them with each other using this as a difference point. In this comparisons we have calculated the costs using simulations as explained. From now on, when we say MDP policy we refer the optimal policy of the M/M/1/K problem and its performance on the M/G/1K version of the problem. We have developed 95% confidence intervals to make the comparisons. For the cases that a confidence interval regarding costs is smaller or greater than each other without any overlap we picked the winner. For the other cases with consists an overlap of the intervals we have not decided to a winner and assumed they are similar in terms of performance.

For the instances with $\rho = 0.75$ we observe in table 4.20 that in 4 of the cases MDP-Informed model outperformed the MDP policy in non-Markovian setting. All of these cases were deterministic instances. On the other hand in 9 of the cases MDP policy outperformed the MDP-Informed model. Eight of these cases were lognormal

and erlang instances and one of them was deterministic. The rest of the five cases has no clear winner. So we can say that MDP-Informed model, without any tuning improved its starting solution in 4 of the cases, for the 9 cases it converged to a worse policy while for the 5 cases it converged a similar quality policy.

Table 4.20. MDP-Informed model's performance in comparison with the MDP policy in Non-Markovian setting with 95% CI for the instances with $\rho = 0.75$

Problem	G	MDP-Low	MDP-High	MDP-InfLow	MDP-InfHigh
3.0-10-0.75-15-2500	Deterministic	98.6594	99.6546	105.8093	111.7987
3.0-10-0.75-15-2500	Erlang	158.0684	158.2816	157.0290	158.8490
3.0-10-0.75-15-2500	Lognormal	173.9404	174.1824	172.9798	175.2562
3.0-10-0.75-5-2500	Deterministic	45.4754	45.8726	40.8402	45.1618
3.0-10-0.75-5-2500	Erlang	57.6122	57.6678	65.7883	66.5237
3.0-10-0.75-5-2500	Lognormal	60.1284	60.2396	75.0540	75.9160
3.0-15-0.75-10-2500	Deterministic	97.4222	98.9298	78.1112	90.4648
3.0-15-0.75-10-2500	Erlang	161.9792	162.1678	161.4218	162.7882
3.0-15-0.75-10-2500	Lognormal	174.7526	174.9994	174.1149	175.4391
3.0-15-0.75-3-2500	Deterministic	42.0938	42.6462	28.2827	32.2353
3.0-15-0.75-3-2500	Erlang	54.0544	54.0956	62.4199	63.0481
3.0-15-0.75-3-2500	Lognormal	55.4793	55.5467	70.5491	71.2929
3.0-25-0.75-1.8-2500	Deterministic	42.1565	43.0875	25.9760	33.7600
3.0-25-0.75-1.8-2500	Erlang	54.9250	54.9670	63.4734	63.9946
3.0-25-0.75-1.8-2500	Lognormal	55.6140	55.7060	70.6719	71.6101
3.0-25-0.75-1.8-5000	Deterministic	42.1565	43.0875	38.7129	44.7671
3.0-25-0.75-1.8-5000	Erlang	54.9250	54.9670	63.4702	63.9858
3.0-25-0.75-1.8-5000	Lognormal	55.6140	55.7060	70.6349	71.5731

For the instances with $\rho = 0.75$ we observe in table 4.21 that in 3 of the cases ordinary Q-Learning model outperformed the MDP policy in non-Markovian settings. All of these cases were deterministic instances. On the other hand in 7 of the cases MDP policy outperformed the MDP-Informed model. All of these cases were lognormal and erlang instances. The rest of the 8 cases has no clear winner. So we can say that ordinary model, without any tuning outperformed MDP in 3 of the cases, for the 7 cases it converged to a worse policy than MDP while for the 8 cases it converged a similar quality policy. It is reasonable that the it improved only deterministic cases, since other service distributions has higher squared coefficient of variation values.

Table 4.21. Ordinary Q-Learning model's performance in comparison with MDP policy in Non-Markovian setting with 95% CI for the instances with $\rho = 0.75$.

Problem	G	MDP-Low	MDP-High	Ordinary-Low	Ordinary-High
3.0-10-0.75-15-2500	Deterministic	98.659	99.655	89.597	99.069
3.0-10-0.75-15-2500	Erlang	158.068	158.282	157.223	159.037
3.0-10-0.75-15-2500	Lognormal	173.940	174.182	173.015	174.679
3.0-10-0.75-5-2500	Deterministic	45.475	45.873	42.859	46.299
3.0-10-0.75-5-2500	Erlang	57.612	57.668	65.712	66.506
3.0-10-0.75-5-2500	Lognormal	60.128	60.240	75.225	76.011
3.0-15-0.75-10-2500	Deterministic	97.422	98.930	77.726	89.916
3.0-15-0.75-10-2500	Erlang	161.979	162.168	161.335	162.749
3.0-15-0.75-10-2500	Lognormal	174.753	174.999	173.976	175.714
3.0-15-0.75-3-2500	Deterministic	42.094	42.646	33.122	39.056
3.0-15-0.75-3-2500	Erlang	54.054	54.096	62.352	63.104
3.0-15-0.75-3-2500	Lognormal	55.479	55.547	70.440	71.384
3.0-25-0.75-1.8-2500	Deterministic	42.157	43.087	27.397	33.325
3.0-25-0.75-1.8-2500	Erlang	54.925	54.967	63.505	63.979
3.0-25-0.75-1.8-2500	Lognormal	55.614	55.706	70.593	71.331
3.0-25-0.75-1.8-5000	Deterministic	42.157	43.087	38.192	44.530
3.0-25-0.75-1.8-5000	Erlang	54.925	54.967	63.455	63.951
3.0-25-0.75-1.8-5000	Lognormal	55.614	55.706	55.472	55.864

For the instances with $\rho = 0.90$ we observe in table 4.22 that in 3 of the cases MDP-Informed model outperformed the MDP results. Two of these cases were deterministic and one of them was lognormal instances. On the other hand in 8 of the cases MDP policy outperformed the MDP-Informed model. All of these cases were lognormal and erlang instances. The rest of the 7 cases has no clear winner. So we can say that MDP-Informed model, without any tuning, improved its starting solution in 3 of the cases, for the 8 cases it converged to a worse policy while for the 7 cases it converged a similar quality policy. It is interesting that it has outperformed the MDP in the problem, 9.0-25-0.9-35-2500.

Table 4.22. MDP-Informed model's performance in comparison with MDP policy in Non-Markovian setting with 95% CI for the instances with $\rho = 0.90$.

Problem	G	MDPLow	MDPHigh	MDP-InfLow	MDP-InfHigh
9.0-10-0.9-45-2500	Deterministic	371.598	373.852	276.151	290.701
9.0-10-0.9-45-2500	Erlang	477.734	478.392	544.552	548.806
9.0-10-0.9-45-2500	Lognormal	531.914	532.598	630.963	637.111
9.0-10-0.9-75-2500	Deterministic	547.626	553.952	539.674	541.812
9.0-10-0.9-75-2500	Erlang	775.808	776.670	819.055	825.735
9.0-10-0.9-75-2500	Lognormal	830.104	830.956	928.440	935.582
9.0-15-0.9-30-2500	Deterministic	361.896	367.598	353.179	373.089
9.0-15-0.9-30-2500	Erlang	494.839	495.163	557.464	560.778
9.0-15-0.9-30-2500	Lognormal	532.943	533.793	635.440	636.882
9.0-15-0.9-30-5000	Deterministic	361.896	367.598	353.806	377.750
9.0-15-0.9-30-5000	Erlang	494.839	495.163	557.464	560.778
9.0-15-0.9-30-5000	Lognormal	532.943	533.793	635.440	636.882
9.0-25-0.9-35-2500	Deterministic	577.736	594.412	549.743	631.661
9.0-25-0.9-35-2500	Erlang	979.235	980.219	979.304	981.950
9.0-25-0.9-35-2500	Lognormal	1059.098	1061.290	954.331	963.525
9.0-25-0.9-35-7500	Deterministic	577.736	594.412	553.203	615.877
9.0-25-0.9-35-7500	Erlang	979.235	980.219	976.022	984.428
9.0-25-0.9-35-7500	Lognormal	1059.098	1061.290	1056.694	1065.100

For the instances with $\rho = 0.90$ we observe in table 4.23 that in 2 of the cases ordinary Q-Learning model outperformed the MDP results. All of these cases were deterministic instances. On the other hand in 7 of the cases MDP policy outperformed the MDP-Informed model. All of these cases were lognormal and erlang instances. The rest of the 9 cases has no clear winner. So we can say that ordinary model, obtain very similar results for $\rho = 0.90$ as well. It outperformed MDP in 2 of the cases, for the 7 cases it converged to a worse policy than MDP while for the 9 cases it converged a similar quality policy.

Table 4.23. Ordinary Q-Learning model's performance in comparison with MDP policy in Non-Markovian setting with 95% CI for the instances with $\rho = 0.90$.

Problem	G	MDP-Low	MDP-High	Ordinary-Low	Ordinary-High
9.0-10-0.9-45-2500	Deterministic	371,598	373,852	274,085	288,737
9.0-10-0.9-45-2500	Erlang	477,734	478,392	544,846	548,866
9.0-10-0.9-45-2500	Lognormal	531,914	532,598	632,372	636,604
9.0-10-0.9-75-2500	Deterministic	547,626	553,952	513,296	533,074
9.0-10-0.9-75-2500	Erlang	775,808	776,670	820,030	824,680
9.0-10-0.9-75-2500	Lognormal	830,104	830,956	929,787	934,987
9.0-15-0.9-30-2500	Deterministic	361,896	367,598	355,496	376,124
9.0-15-0.9-30-2500	Erlang	494,839	495,163	557,234	560,422
9.0-15-0.9-30-2500	Lognormal	532,943	533,793	532,394	535,966
9.0-15-0.9-30-5000	Deterministic	361,896	367,598	356,371	383,409
9.0-15-0.9-30-5000	Erlang	494,839	495,163	557,759	560,171
9.0-15-0.9-30-5000	Lognormal	532,943	533,793	633,874	636,756
9.0-25-0.9-35-2500	Deterministic	577,736	594,412	554,322	622,630
9.0-25-0.9-35-2500	Erlang	979,235	980,219	980,130	983,602
9.0-25-0.9-35-2500	Lognormal	1059,098	1061,290	1053,777	1066,417
9.0-25-0.9-35-7500	Deterministic	577,736	594,412	557,639	615,449
9.0-25-0.9-35-7500	Erlang	979,235	980,219	980,188	983,502
9.0-25-0.9-35-7500	Lognormal	1059,098	1061,290	1053,476	1065,124

For the instances with $\rho = 0.95$ we observe in table 4.24 that in 3 of the cases MDP-Informed model outperformed the MDP results. All of these cases were deterministic. On the other hand in 11 of the cases MDP policy outperformed the MDP-Informed model. All of these cases were lognormal and erlang instances. The rest of the 4 cases has no clear winner. So we can say that MDP-Informed model, without any tuning, improved its starting solution in 3 of the cases, for the 11 cases it converged to a worse policy while for the 4 cases it converged a similar quality policy. It performed worse in this instances compared to its performance on others.

Table 4.24. MDP-Informed Model's performance in comparison with MDP policy in Non-Markovian Setting with 95% CI for the instances with $\rho = 0.95$.

Problem	G	MDPLow	MDPHigh	MDP-InfLow	MDP-InfHigh
19.0-10-0.95-375-2500	Deterministic	2728.356	2729.126	2357.662	2446.278
19.0-10-0.95-375-2500	Erlang	3750.054	3753.792	3972.563	3993.515
19.0-10-0.95-375-2500	Lognormal	4107.120	4109.050	4442.039	4603.265
19.0-10-0.95-375-5000	Deterministic	2728.356	2729.126	2357.662	2446.278
19.0-10-0.95-375-5000	Erlang	3750.054	3753.792	3972.563	3993.515
19.0-10-0.95-375-5000	Lognormal	4107.120	4109.050	4502.657	4522.851
19.0-15-0.95-185-2500	Deterministic	2092.494	2115.054	1968.741	2056.579
19.0-15-0.95-185-2500	Erlang	2895.529	2897.555	3156.248	3168.240
19.0-15-0.95-185-2500	Lognormal	3137.276	3142.494	3537.134	3551.912
19.0-15-0.95-185-5000	Deterministic	2092.494	2115.054	2135.341	2251.309
19.0-15-0.95-185-5000	Erlang	2895.529	2897.555	3173.414	3181.430
19.0-15-0.95-185-5000	Lognormal	3137.276	3142.494	3536.059	3553.201
19.0-25-0.95-70-2500	Deterministic	1452.153	1479.645	1391.192	1485.150
19.0-25-0.95-70-2500	Erlang	1950.857	1951.905	2226.665	2233.161
19.0-25-0.95-70-2500	Lognormal	2114.631	2117.305	2111.274	2125.286
19.0-25-0.95-70-5000	Deterministic	1452.153	1479.645	1385.899	1485.919
19.0-25-0.95-70-5000	Erlang	1950.857	1951.905	2226.438	2232.030
19.0-25-0.95-70-5000	Lognormal	2114.631	2117.305	2113.595	2120.249

For the instances with $\rho = 0.95$ we observe in table 4.25 that in 3 of the cases ordinary Q-Learning model outperformed the MDP results. All of these cases were deterministic instances. On the other hand in 11 of the cases MDP policy outperformed the MDP-Informed model. All of these cases were lognormal and erlang instances. The rest of the 4 cases has no clear winner. So we can say that ordinary model, obtain very similar results to MDP-informed model for the $\rho = 0.95$. It outperformed MDP in 3 of the cases, for the 11 cases it converged to a worse policy than MDP while for the 4 cases it converged a similar quality policy. We can say both models performed the worse for the problems with $\rho = 0.95$

Table 4.25. Ordinary Q-Learning Model's performance in comparison with MDP policy in Non-Markovian setting with 95% CI for the instances with $\rho = 0.95$.

Problem	G	MDPLow	MDPHigh	Q-Learn Low	Q-LearnHigh
19.0-10-0.95-375-2500	Deterministic	2728.356	2729.126	2357.662	2446.278
19.0-10-0.95-375-2500	Erlang	3750.054	3753.792	3806.032	3820.720
19.0-10-0.95-375-2500	Lognormal	4107.120	4109.050	4481.253	4544.139
19.0-10-0.95-375-5000	Deterministic	2728.356	2729.126	2357.662	2446.278
19.0-10-0.95-375-5000	Erlang	3750.054	3753.792	3804.719	3819.203
19.0-10-0.95-375-5000	Lognormal	4107.120	4109.050	4491.495	4542.745
19.0-15-0.95-185-2500	Deterministic	2092.494	2115.054	1903.370	1977.370
19.0-15-0.95-185-2500	Erlang	2895.529	2897.555	3158.151	3166.889
19.0-15-0.95-185-2500	Lognormal	3137.276	3142.494	3527.735	3563.943
19.0-15-0.95-185-5000	Deterministic	2092.494	2115.054	2002.598	2096.002
19.0-15-0.95-185-5000	Erlang	2895.529	2897.555	3158.765	3166.241
19.0-15-0.95-185-5000	Lognormal	3137.276	3142.494	3119.689	3152.399
19.0-25-0.95-70-2500	Deterministic	1452.153	1479.645	1387.522	1479.604
19.0-25-0.95-70-2500	Erlang	1950.857	1951.905	2226.108	2232.496
19.0-25-0.95-70-2500	Lognormal	2114.631	2117.305	2520.960	2527.092
19.0-25-0.95-70-5000	Deterministic	1452.153	1479.645	1390.732	1484.296
19.0-25-0.95-70-5000	Erlang	1950.857	1951.905	2227.515	2233.911
19.0-25-0.95-70-5000	Lognormal	2114.631	2117.305	2517.260	2529.824

To summarize from all of these trials with the M/G/1/K case we see that both of models seem to adapt to the deterministic setting well, without any tuning made to the hyperparameters based on the fact that they either outperform or perform similarly to MDP policy in general. However, as the squared coefficient of variation of the distribution increases they cannot adapt that well and perform worse than the MDP policy. Yet we have to realize that the MDP policy which is optimal for M/M/1/K problems and sub-optimal for the M/G/1/K problems has been obtained with the real values of the arrival and service rates. We compare our models with an all-knowing ideal solution, and for deterministic cases it can beat it without any tuning is necessary. This also shows the importance of the effect of the hyperparameters which orchestrates the exploration-exploitation balance for Q-Learning algorithm. This results show us that the adaptability of the Q-Learning algorithm tied to a good exploration-exploitation balance.

Now we will continue with the comparison of the performances of the MDP-Informed model and ordinary Q-Learning model using the average expected cost metric estimated in simulations and the number of iterations required to converge. We will examine the confidence intervals of the 2 different Q-Learning variations in detail.

Table 4.26 show us that only in the instances 3.0-25-0.75-1,8-5000 with lognormal service, 3.0-10-0.75-15-2500 with deterministic service and 3.0-15-0.75-3-2500 with deterministic service there is a clear winner. In the first two instances the ordinary model wins while in the last instance MDP-Informed model wins, considering 95% confidence intervals. For the other instances the confidence intervals overlap each other so we cannot decide a winner there. We observe in table 4.27, in terms of iterations required to converge MDP-Informed model needed less iterations in general. Except the Erlang instance on the problem 3.0-15-0.75-10-2500 MDP-Informed model always converged in less iterations.

Table 4.26. Comparison of performance for MDP-Informed models and ordinary models with 95% Confidence Intervals and $\rho = 0.75$

Problem	G	MDP-InfLow	MDP-InfHigh	OrdinaryLow	OrdinaryHigh
3.0-10-0.75-15-2500	Deterministic	105.809	111.799	89.597	99.069
3.0-10-0.75-15-2500	Erlang	157.029	158.849	157.223	159.037
3.0-10-0.75-15-2500	Lognormal	172.980	175.256	173.015	174.679
3.0-10-0.75-5-2500	Deterministic	40.840	45.162	42.859	46.299
3.0-10-0.75-5-2500	Erlang	65.788	66.524	65.712	66.506
3.0-10-0.75-5-2500	Lognormal	75.054	75.916	75.225	76.011
3.0-15-0.75-10-2500	Deterministic	78.111	90.465	77.726	89.916
3.0-15-0.75-10-2500	Erlang	161.422	162.788	161.335	162.749
3.0-15-0.75-10-2500	Lognormal	174.115	175.439	173.976	175.714
3.0-15-0.75-3-2500	Deterministic	28.283	32.235	33.122	39.056
3.0-15-0.75-3-2500	Erlang	62.420	63.048	62.352	63.104
3.0-15-0.75-3-2500	Lognormal	70.549	71.293	70.440	71.384
3.0-25-0.75-1.8-2500	Deterministic	25.976	33.760	27.397	33.325
3.0-25-0.75-1.8-2500	Erlang	63.473	63.995	63.505	63.979
3.0-25-0.75-1.8-2500	Lognormal	70.672	71.610	70.593	71.331
3.0-25-0.75-1.8-5000	Deterministic	38.713	44.767	38.192	44.530
3.0-25-0.75-1.8-5000	Erlang	63.470	63.986	63.455	63.951
3.0-25-0.75-1.8-5000	Lognormal	70.635	71.573	55.472	55.864

Table 4.27. Comparison of cost differences and iterations for MDP-Informed models and ordinary models for the instances with $\rho = 0.75$

Problem	G	Ordinary-MDPInformed	MDP-InformedIter	OrdinaryIter
3.0-10-0.75-15-2500	Deterministic	-14.471	11	22
3.0-10-0.75-15-2500	Erlang	0.191	25	29
3.0-10-0.75-15-2500	Lognormal	-0.271	37	35
3.0-10-0.75-5-2500	Deterministic	1.578	23	24
3.0-10-0.75-5-2500	Erlang	-0.047	27	37
3.0-10-0.75-5-2500	Lognormal	0.133	39	37
3.0-15-0.75-10-2500	Deterministic	-0.467	74	78
3.0-15-0.75-10-2500	Erlang	-0.063	439	283
3.0-15-0.75-10-2500	Lognormal	0.068	484	630
3.0-15-0.75-3-2500	Deterministic	5.830	25	29
3.0-15-0.75-3-2500	Erlang	-0.006	75	90
3.0-15-0.75-3-2500	Lognormal	-0.009	80	87
3.0-25-0.75-1,8-2500	Deterministic	0.493	43	46
3.0-25-0.75-1,8-2500	Erlang	0.008	90	104
3.0-25-0.75-1,8-2500	Lognormal	-0.179	92	96
3.0-25-0.75-1,8-5000	Deterministic	-0.379	48	46
3.0-25-0.75-1,8-5000	Erlang	-0.025	48	57
3.0-25-0.75-1,8-5000	Lognormal	-15.436	51	49

The table 4.28 show us that only in the instances 9.0-10-0.9-75-2500 with deterministic service, 9.0-15-0.9-30-2500 with lognormal service and 9.0-25-0.9-35-2500 with lognormal service there is a clear winner. In the first two instances the ordinary model wins while in the last instance MDP-Informed model wins, considering 95% confidence intervals. For the other instances the confidence intervals overlap each other so we cannot decide a winner there. We observe in table 4.29, in terms of iterations required to converge MDP-Informed model in 10 of the 18 instances MDP-Informed needed less iterations and in the rest of the instances the ordinary model were faster. There is no clear winner in terms of convergence speed for these instances.

Table 4.28. Comparison of performance for MDP-Informed models and ordinary models with 95% Confidence Intervals and $\rho = 0.90$

Problem	G	MDP-InfLow	MDP-InfHigh	OrdinaryLow	OrdinaryHigh
9.0-10-0.9-45-2500	Deterministic	276.1513	290.7007	274.0850	288.7370
9.0-10-0.9-45-2500	Erlang	544.5521	548.8059	544.8459	548.8661
9.0-10-0.9-45-2500	Lognormal	630.9635	637.1105	632.3725	636.6035
9.0-10-0.9-75-2500	Deterministic	539.6738	541.8122	513.2958	533.0742
9.0-10-0.9-75-2500	Erlang	819.0549	825.7351	820.0299	824.6801
9.0-10-0.9-75-2500	Lognormal	928.4400	935.5820	929.7868	934.9872
9.0-15-0.9-30-2500	Deterministic	353.1793	373.0887	355.4960	376.1240
9.0-15-0.9-30-2500	Erlang	557.4645	560.7775	557.2342	560.4218
9.0-15-0.9-30-2500	Lognormal	635.4397	636.8823	532.3938	535.9662
9.0-15-0.9-30-5000	Deterministic	353.8061	377.7499	356.3713	383.4087
9.0-15-0.9-30-5000	Erlang	557.4645	560.7775	557.7590	560.1710
9.0-15-0.9-30-5000	Lognormal	635.4397	636.8823	633.8738	636.7562
9.0-25-0.9-35-2500	Deterministic	549.7426	631.6614	554.3222	622.6298
9.0-25-0.9-35-2500	Erlang	979.3041	981.9499	980.1297	983.6023
9.0-25-0.9-35-2500	Lognormal	954.3305	963.5255	1053.7775	1066.4165
9.0-25-0.9-35-7500	Deterministic	553.2032	615.8768	557.6392	615.4488
9.0-25-0.9-35-7500	Erlang	976.0224	984.4276	980.1885	983.5015
9.0-25-0.9-35-7500	Lognormal	1056.6944	1065.0996	1053.4760	1065.1240

Table 4.29. Comparison of cost differences and iterations for MDP-Informed models and ordinary models for the instances with $\rho = 0.90$

Problem	G	Ordinary-MDPInformed	MDP-InformedIter	OrdinaryIter
9.0-10-0.9-45-2500	Deterministic	-2.015	36	31
9.0-10-0.9-45-2500	Erlang	0.177	273	213
9.0-10-0.9-45-2500	Lognormal	0.451	261	172
9.0-10-0.9-75-2500	Deterministic	-17.558	24	31
9.0-10-0.9-75-2500	Erlang	-0.040	54	59
9.0-10-0.9-75-2500	Lognormal	0.376	93	94
9.0-15-0.9-30-2500	Deterministic	2.676	20	18
9.0-15-0.9-30-2500	Erlang	-0.293	60	74
9.0-15-0.9-30-2500	Lognormal	-101.981	76	90
9.0-15-0.9-30-5000	Deterministic	4.112	17	16
9.0-15-0.9-30-5000	Erlang	-0.156	47	38
9.0-15-0.9-30-5000	Lognormal	-0.846	75	66
9.0-25-0.9-35-2500	Deterministic	-2.226	16	24
9.0-25-0.9-35-2500	Erlang	1.239	84	77
9.0-25-0.9-35-2500	Lognormal	101.169	152	65
9.0-25-0.9-35-7500	Deterministic	2.004	11	19
9.0-25-0.9-35-7500	Erlang	1.620	51	70
9.0-25-0.9-35-7500	Lognormal	-1.597	176	159

The table 4.30 show us more clear results than the other tables. In 5 of the instances the ordinary model performed better which are, 19.0-10-0.95-375-2500 with erlang service, 19.0-10-0.95-375-5000 with erlang service and 19.0-15-0.95-185-5000 with all three services. In 2 of the instances MDP-Informed model performed which are 19.0-25-0.95-70-2500 with lognormal service and 19.0-25-0.95-70-5000 with lognormal service. For the other instances the confidence intervals overlap each other so we cannot decide a winner there. We observe in table 4.31, in terms of iterations required to converge MDP-Informed model in 5 of the 18 instances MDP-Informed needed less iterations and in the rest of the instances the ordinary model were faster. It seems for these instances the ordinary model were quicker to converge.

Table 4.30. Comparison of performance for MDP-Informed models and ordinary models with 95% Confidence Intervals and $\rho = 0.95$

Problem	G	MDP-InfLow	MDP-InfHigh	OrdinaryLow	OrdinaryHigh
19.0-10-0.95-375-2500	Deterministic	2357.662	2446.278	2357.662	2446.278
19.0-10-0.95-375-2500	Erlang	3972.563	3993.515	3806.032	3820.720
19.0-10-0.95-375-2500	Lognormal	4442.039	4603.265	4481.253	4544.139
19.0-10-0.95-375-5000	Deterministic	2357.662	2446.278	2357.662	2446.278
19.0-10-0.95-375-5000	Erlang	3972.563	3993.515	3804.719	3819.203
19.0-10-0.95-375-5000	Lognormal	4502.657	4522.851	4491.495	4542.745
19.0-15-0.95-185-2500	Deterministic	1968.741	2056.579	1903.370	1977.370
19.0-15-0.95-185-2500	Erlang	3156.248	3168.240	3158.151	3166.889
19.0-15-0.95-185-2500	Lognormal	3537.134	3551.912	3527.735	3563.943
19.0-15-0.95-185-5000	Deterministic	2135.341	2251.309	2002.598	2096.002
19.0-15-0.95-185-5000	Erlang	3173.414	3181.430	3158.765	3166.241
19.0-15-0.95-185-5000	Lognormal	3536.059	3553.201	3119.689	3152.399
19.0-25-0.95-70-2500	Deterministic	1391.192	1485.150	1387.522	1479.604
19.0-25-0.95-70-2500	Erlang	2226.665	2233.161	2226.108	2232.496
19.0-25-0.95-70-2500	Lognormal	2111.274	2125.286	2520.960	2527.092
19.0-25-0.95-70-5000	Deterministic	1385.899	1485.919	1390.732	1484.296
19.0-25-0.95-70-5000	Erlang	2226.438	2232.030	2227.515	2233.911
19.0-25-0.95-70-5000	Lognormal	2113.595	2120.249	2517.260	2529.824

Table 4.31. Comparison of cost differences and iterations for MDP-Informed models and ordinary models for the instances with $\rho = 0.95$

Problem	G	Ordinary-MDPInformed	MDP-InformedIter	OrdinaryIter
19.0-10-0.95-375-2500	Deterministic	0.000	32	130
19.0-10-0.95-375-2500	Erlang	-169.663	204	73
19.0-10-0.95-375-2500	Lognormal	-9.956	67	99
19.0-10-0.95-375-5000	Deterministic	0.000	75	69
19.0-10-0.95-375-5000	Erlang	-171.078	226	170
19.0-10-0.95-375-5000	Lognormal	4.366	202	85
19.0-15-0.95-185-2500	Deterministic	-72.290	26	34
19.0-15-0.95-185-2500	Erlang	0.276	105	75
19.0-15-0.95-185-2500	Lognormal	1.316	114	86
19.0-15-0.95-185-5000	Deterministic	-144.025	122	79
19.0-15-0.95-185-5000	Erlang	-14.919	456	472
19.0-15-0.95-185-5000	Lognormal	-408.586	846	435
19.0-25-0.95-70-2500	Deterministic	-4.608	66	67
19.0-25-0.95-70-2500	Erlang	-0.611	121	121
19.0-25-0.95-70-2500	Lognormal	405.746	143	118
19.0-25-0.95-70-5000	Deterministic	1.605	153	148
19.0-25-0.95-70-5000	Erlang	1.479	469	395
19.0-25-0.95-70-5000	Lognormal	406.620	628	379

When we combine the detailed comparisons we see that only 13 of the 54 total instances there exist a clear winner. In 9 of those cases the ordinary model were better while on the 4 of them the MDP-Informed model outperformed the ordinary model. In terms of number of iterations required to converge we see that in instances with $\rho = 0.75$ the MDP-Informed models were quicker, on the other hand in instances with $\rho = 0.95$ the ordinary models were quicker. For the instances with $\rho = 0.90$ we see a tie in terms of convergence speeds. In the end we do not think that there is clear winner among the MDP-Informed model and the ordinary model. In fact the ordinary model has more wins against the MDP-Informed model among the cases that we can decide. This result demonstrates that, the MDP-Informed model did not improve the ordinary model. However seeing that it improved in some of the fewer cases might tell us that to research this question further by examining that cases in detail. We also believe that hyperparameter values may have affected the result. Using hyperparameter tuned models may create different results. Yet we compared both of the models in equal

settings with the same hyperparameters and this may let us to conclude that MDP-Informed models do not necessarily improve the ordinary model.

4.4. Overall Findings and Insights

The analysis and experimentation conducted in this thesis yield several significant findings regarding the application of reinforcement learning (RL), specifically Q-learning, to queueing control problems. These insights derived from the outcomes of numerous simulations and comparisons with the MDP results.

Regarding the first objective of the thesis we saw that the Q-Learning can effectively and efficiently learn and optimize the M/M/1/K problem from scratch. This study explored the impact of 3 different exploration strategies of ϵ -Greedy, Boltzmann exploration, and Upper Confidence Bound (UCB) on the convergence and performance of the Q-learning algorithm. It has been shown that the best performer among the three strategies was the Boltzman Strategy which consistently performed better in many of the experiment instances compared to others and created results that are very close to optimal in general. It was the only strategy that reached the optimal strategy in some of the instances. The second place was of the ϵ -Greedy Strategy which performed well and in some instances beat the Boltzman Strategy. The UCB strategy performed worst among the three strategies and in some instances it created results with high deviations from the optimal policy. The sensitivity of Q-learning algorithm to hyperparameters was evident throughout the experiments with all three strategies. Incorrect hyperparameter settings led to prolonged convergence times and suboptimal performance. In this thesis we tried in total of 864 different instances and hyperparameter values to pick the among them. The all three versions of the algorithm was very sensitive to the hyperparameter values. This emphasizes the necessity for careful tuning and potential use of automated hyperparameter optimization techniques in practical applications. Another remark is that in our observations increasing the episode lengths did not create an advantage because surprisingly it did not improve solution quality in most of the experimented instances.

For the second objective of this thesis in our tests with the M/G/1/K problems we observed that it can adapt easily and without any tuning to the deterministic service rates. However it had difficulties to adapt for the erlang and lognormal service distributions. Yet considering there was hyperparameter tuning is done for the M/G/1/K case, this results show us that Q-Learning can be efficient and effective in practice and can adapt to the changes to some degree even without any tuning is done. To make it fully adaptive to the changes, a mechanism that automatically tunes the parameters should be established. Establishing such a mechanism should be a further research direction. Because we believe that if tuning problem is solved, Q-Learning can be strong alternative for any of the techniques that uses predict then optimize approach. As the second part of the second objective in our tests we saw that, embedding the model knowledge to the Q-Learning algorithm by using the MDP policy as a starting point did not create good any improvement to the model in our cases. Yet we cannot make a generalization of this result because of two reasons. First we did not make a hyperparameter tuning, maybe the comparison of well-tuned models create results that justify the effectiveness of the MDP-Informed model. And as the second matter one can try to embed model information with different techniques, seeing that our method did not work cannot prove that this is the case for all the alternatives. This topic requires further research on the topic of adding model knowledge to the model-free approaches.

5. CONCLUSION

Our first objective in this thesis was to compare the efficiency and effectiveness of the RL algorithms in the M/M/1/K service rate balancing problem with the results of MDP. The second objective was to extend the queueing model by considering non-Markovian service rates and use the RL method to solve the problem. For the extension case our purpose was to study the benefits of using MDP results as an initial starting solution to observe whether the Q-Learning algorithm would improve the starting point.

Accordingly this thesis has investigated the application of reinforcement learning (RL), specifically Q-learning, to the optimization of M/M/1/K service rate balancing problem. Through the examination of M/M/1/K and its extension to M/G/1/K queueing system, this study has exhibited the feasibility and possible benefits of employing reinforcement learning techniques against the traditional approach of using MDP algorithms on estimated parameters. The study has demonstrated that Q-learning is capable of optimizing service rates in M/M/1/K queueing systems in an efficient manner from scratch, reaching performance levels that are very similar to those obtained with the MDP optimal model which has the knowledge of the true parameters. This demonstrates that Q-learning is a useful tool for queueing control, especially in real life situations where actual parameter values are unknown. By examining the impact of different exploration strategies ϵ -Greedy, Boltzmann exploration, and Upper Confidence Bound (UCB), this research has identified the strengths and weaknesses of each approach. The Boltzmann strategy emerged as the most effective in balancing exploration and exploitation, leading to a fast convergence and good quality results. The results in numerical analysis emphasize how important hyperparameter tuning is to Q-learning's effectiveness by trying 864 different instances and showing the best among them. The algorithm's performance can be severely hindered by incorrect settings, which emphasizes the necessity of automated optimization methods or careful tuning in real-world applications. This also shows itself in Q-learning's extension to M/G/1/K queueing systems with different service time distributions. The experiments on the extension

case shows that the algorithm adapts to the changes to some degree even without any tuning is done however to obtain a model that can adapt to any service distribution one should create a mechanism that adaptively tunes the hyperparameters as well. The insights gained from this research suggest that Q-learning, with appropriate exploration strategies and hyperparameter tuning, can be a powerful tool for queueing control in diverse settings. Future work should focus on developing more robust hyperparameter optimization methods, reducing data needs, handling the learning the rare cases and more complex versions of queueing problems.

In conclusion, this thesis has demonstrated that reinforcement learning, particularly Q-learning, can effectively and efficiently overcome well-known queueing control problems, offering a flexible and adaptable alternative to traditional methods. The findings contribute to the growing body of knowledge on applying RL to operational research and provide a foundation for further research and practical applications in this area. We can open up new possibilities for optimizing complex systems and enhancing operational efficiency across a range of domains by carrying out further research and development on RL techniques.

REFERENCES

1. Gross, D., J. Shortie, J. Thompsonm and C. Harris, *Fundamentals of Queueing Theory Fourth Edition*, Hoboken, New Jersey, 2008.
2. Govil, M. K. and M. C. Fu, “Queueing Theory in Manufacturing: A Survey”, *Journal of manufacturing systems*, Vol. 18, No. 3, pp. 214–240, 1999.
3. Chauhan, S., R. Tiwari and M. L. Saini, “Applications of Queueing Theory in Cloud Computing: A Review”, *Kilby*, Vol. 100, p. 7th, 2023.
4. Vilaplana, J., F. Solsona, I. Teixidó, J. Mateo, F. Abella and J. Rius, “A Queueing Theory Model for Cloud Computing”, *The Journal of Supercomputing*, Vol. 69, pp. 492–507, 2014.
5. Ghimire, S., G. B. Thapa, R. Ghinire and S. Silvestrov, “A Survey on Queueing Systems with Mathematical Models and Applications”, *American Journal of Operation Research*, Vol. 7, No. 1, pp. 1–14, 2017.
6. Fomundam, S. and J. W. Herrmann, “A Survey of Queueing Theory Applications in Healthcare”, *The Institute for Systems Research-Isr Technical Report*, Vol. 24, No. 2007-24, 2007.
7. MacGregor Smith, J. and F. R. Cruz, “The Buffer Allocation Problem for General Finite Buffer Queueing Networks”, *IIE transactions*, Vol. 37, No. 4, pp. 343–365, 2005.
8. Ross, S. M., *Introduction to Probability Models*, Academic press, The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, UK, 2014.
9. Smith, J. M., “Optimal Design and Performance Modelling of M/G/1/K Queueing Systems”, *Mathematical and Computer Modelling*, Vol. 39, No. 9-10, pp. 1049–

1081, 2004.

10. Smith, J. M., “M/G/c/K Blocking Probability Models and System Performance”, *Performance Evaluation*, Vol. 52, No. 4, pp. 237–267, 2003.
11. George, J. M. and J. M. Harrison, “Dynamic Control of A Queue with Adjustable Service Rate”, *Operations research*, Vol. 49, No. 5, pp. 720–731, 2001.
12. Adusumilli, K. M. and J. J. Hasenbein, “Dynamic Admission and Service Rate Control of A Queue”, *Queueing Systems*, Vol. 66, No. 2, pp. 131–154, 2010.
13. Khayyati, S. and B. Tan, “A Machine Learning Approach for Implementing Data-Driven Production Control Policies”, *International Journal of Production Research*, Vol. 60, No. 10, pp. 3107–3128, 2022.
14. Khayyati, S. and B. Tan, “Supervised-Learning-Based Approximation Method for Multi-server Queueing Networks Under Different Service Disciplines with Correlated Interarrival and Service Times”, *International Journal of Production Research*, Vol. 60, No. 17, pp. 5176–5200, 2022.
15. Kyritsis, A. I. and M. Deriaz, “A Machine Learning Approach to Waiting Time Prediction in Queueing Scenarios”, *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*, pp. 17–21, IEEE, Laguna Hills, CA, USA, 2019.
16. Hermanto, R. P. S., A. Nugroho *et al.*, “Waiting-Time Estimation in Bank Customer Queues using RPROP Neural Networks”, *Procedia Computer Science*, Vol. 135, pp. 35–42, 2018.
17. L’Ecuyer, P., N. Giroux and P. W. Glynn, “Stochastic Optimization by Simulation: Numerical Experiments with the M/M/1 Queue in Steady-state”, *Management science*, Vol. 40, No. 10, pp. 1245–1261, 1994.

18. Bertsekas, D., *A Course in Reinforcement Learning*, Athena Scientific, Belmont, Massachusetts, 2023.
19. Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT press, Cambridge, Massachusetts London, England, 2018.
20. Ishii, S., W. Yoshida and J. Yoshimoto, “Control of Exploitation–Exploration Meta-Parameter in Reinforcement Learning”, *Neural Networks*, Vol. 15, No. 4-6, pp. 665–687, 2002.
21. Guo, M., Y. Liu and J. Malec, “A New Q-learning Algorithm Based on the Metropolis Criterion”, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 34, No. 5, pp. 2140–2143, 2004.
22. Leonardos, S., G. Piliouras and K. Spendlove, “Exploration-Exploitation in Multi-Agent Competition: Convergence with Bounded Rationality”, *Advances in Neural Information Processing Systems*, Vol. 34, pp. 26318–26331, 2021.
23. Schaul, T., J. Quan, I. Antonoglou and D. Silver, “Prioritized Experience Replay”, *arXiv preprint arXiv:1511.05952*, 2015.
24. Liu, B., Q. Xie and E. Modiano, “Reinforcement Learning for Optimal Control of Queueing Systems”, *2019 57th annual allerton conference on communication, control, and computing (allerton)*, pp. 663–670, IEEE, Monticello, IL, USA, 2019.
25. Dai, J. G. and M. Gluzman, “Queueing Network Controls via Deep Reinforcement Learning”, *Stochastic Systems*, Vol. 12, No. 1, pp. 30–67, 2022.
26. Bachl, M., J. Fabini and T. Zseby, “LFQ: Online Learning of Per-Flow Queuing Policies Using Deep Reinforcement Learning”, *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pp. 417–420, IEEE, Sydney, NSW, Australia, 2020.
27. Raeis, M., A. Tizghadam and A. Leon-Garcia, “Queue-learning: A Reinforcement

- Learning Approach for Providing Quality of Service”, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35, pp. 461–468, Vancouver, Canada, 2021.
28. Xu, Z., J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu and D. Yang, “Experience-Driven Networking: A Deep Reinforcement Learning Based Approach”, *IEEE INFOCOM 2018-IEEE conference on computer communications*, pp. 1871–1879, IEEE, Honolulu, HI, USA, 2018.
 29. Kim, M., M. Jaseemuddin and A. Anpalagan, “Deep Reinforcement Learning Based Active Queue Management for Iot Networks”, *Journal of Network and Systems Management*, Vol. 29, No. 3, p. 34, 2021.
 30. Hsu, W.-K., J. Xu, X. Lin and M. R. Bell, “Integrated Online Learning and Adaptive Control in Queueing Systems with Uncertain Payoffs”, *Operations Research*, Vol. 70, No. 2, pp. 1166–1181, 2022.
 31. Sentenac, F., E. Boursier and V. Perchet, “Decentralized Learning in Online Queueing Systems”, *Advances in Neural Information Processing Systems*, Vol. 34, pp. 18501–18512, 2021.
 32. Carner, J., A. Mestres, E. Alarcón and A. Cabellos, “Machine Learning-based Network Modeling: An Artificial Neural Network Model vs a Theoretical Inspired Model”, *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 522–524, IEEE, Milan, Italy, 2017.
 33. Chen, X., Y. Liu and G. Hong, “Online learning and Optimization for Queues with Unknown Demand Curve and Service Distribution”, *arXiv preprint arXiv:2303.03399*, 2023.
 34. Chen, X., Y. Liu and G. Hong, “An Online Learning Approach to Dynamic Pricing and Capacity Sizing in Service Systems”, *arXiv:2009.02911*, 2022,

<https://arxiv.org/abs/2009.02911>.

35. Tran, T. H., L. M. Nguyen and K. Scheinberg, “Finding Optimal Policy for Queueing Models: New Parameterization”, *arXiv preprint arXiv:2206.10073*, 2022.
36. Chen, W., *Online Learning Algorithms for Stochastic Inventory and Queueing Systems*, Ph.D. Thesis, University of Michigan, 2019.
37. Jia, H., C. Shi and S. Shen, “Online Learning and Pricing with Reusable Resources: Linear Bandits with Sub-Exponential Rewards”, *International Conference on Machine Learning*, pp. 10135–10160, PMLR, Baltimore, Maryland, USA,, 2022.
38. Krishnasamy, S., R. Sen, R. Johari and S. Shakkottai, “Learning Unknown Service Rates in Queues: A Multiarmed Bandit Approach”, *Operations research*, Vol. 69, No. 1, pp. 315–330, 2021.
39. Zhong, Y., J. R. Birge and A. Ward, “Learning the Scheduling Policy in Time-varying Multiclass Many Server Queues with Abandonment”, *Available at SSRN 4090021*, 2022.
40. Shah, D., Q. Xie and Z. Xu, “Stable Reinforcement Learning with Unbounded State Space”, *arXiv preprint arXiv:2006.04353*, 2020.
41. Watkins, C. J. C. H., *Learning from Delayed Rewards*, Ph.D. Thesis, King’s College, Cambridge United Kingdom, 1989.
42. Puterman, M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Hoboken, New Jersey, 1994.
43. Watkins, C. J. and P. Dayan, “Q-learning”, *Machine learning*, Vol. 8, pp. 279–292, 1992.
44. Szepesvári, C., “The Asymptotic Convergence-Rate of Q-learning”, *Advances in*

Neural Information Processing Systems, Vol. 10, pp. 1064–1070.

45. Lyu, Y., A. Côme, Y. Zhang and M. S. Talebi, “Scaling Up Q-learning via Exploiting State–Action Equivalence”, *Entropy*, Vol. 25, No. 4, p. 584, 2023.
46. McFarlane, R., “A Survey of Exploration Strategies in Reinforcement Learning”, *McGill University*, Vol. 3, pp. 17–18, 2018.
47. Ye, W. and D. Chen, “Analysis of Performance Measure in Q Learning with UCB Exploration”, *Mathematics*, Vol. 10, No. 4, p. 575, 2022.
48. Tijssma, A. D., M. M. Drugan and M. A. Wiering, “Comparing Exploration Strategies for Q-learning in Random Stochastic Mazes”, *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, IEEE, Athens, Greece, 2016.
49. Lin, L.-J., “Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching”, *Machine Learning*, Vol. 8, pp. 293–321, 1992.
50. Asadi, K. and M. L. Littman, “An Alternative Softmax Operator for Reinforcement Learning”, *International Conference on Machine Learning*, pp. 243–252, PMLR, Sydney, Australia, 2017.

APPENDIX A: TABLES REGARDING SIMULATION VERIFICATION

Here lies the results of the verification trials of the M/M/1 simulations using $\rho = 0.25, 0.5, 0.75, 0.8$ and 0.9 together with the verification trials of M/M/1/K simulations with $K = 10, 15$ and 25 .

Table A.1. Verification of Parameters M/M/1 Queue with $\rho = 0.25$ and 95%
Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 95%	CI Upper 95%
L	0.3333	0.3349	0.0110	0.3318	0.3381
L_q	0.0833	0.0842	0.0072	0.0822	0.0863
ω	0.3333	0.3346	0.0088	0.3320	0.3371
ω_q	0.0833	0.0841	0.0067	0.0822	0.0860
λ	1.0000	1.0011	0.0160	0.9965	1.0056
μ	4.0000	3.9933	0.0588	3.9766	4.0100
ρ	0.2500	0.2507	0.0052	0.2492	0.2522

Table A.2. Verification of Parameters M/M/1 Queue with $\rho = 0.25$ and 90%
Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
L	0.3333	0.3349	0.0110	0.3323	0.3376
L_q	0.0833	0.0842	0.0072	0.0825	0.0859
ω	0.3333	0.3346	0.0088	0.3325	0.3367
ω_q	0.0833	0.0841	0.0067	0.0825	0.0857
λ	1.0000	1.0011	0.0160	0.9973	1.0049
μ	4.0000	3.9933	0.0588	3.9793	4.0072
ρ	0.2500	0.2507	0.0052	0.2495	0.2520

Table A.3. Verification of Parameters M/M/1 Queue with $\rho = 0.5$ and 95% Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 95%	CI Upper 95%
L	1.0000	1.0085	0.0444	0.9959	1.0211
L_q	0.5000	0.5082	0.0372	0.4976	0.5187
ω	0.5000	0.5036	0.0189	0.4982	0.5089
ω_q	0.2500	0.2537	0.0169	0.2488	0.2585
λ	2.0000	2.0021	0.0253	1.9949	2.0093
μ	4.0000	4.0019	0.0509	3.9874	4.0163
ρ	0.5000	0.5003	0.0088	0.4978	0.5029

Table A.4. Verification of Parameters M/M/1 Queue with $\rho = 0.5$ and 90% Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
L	1.0000	1.0085	0.0444	0.9980	1.0190
L_q	0.5000	0.5082	0.0372	0.4993	0.5170
ω	0.5000	0.5036	0.0189	0.4991	0.5080
ω_q	0.2500	0.2537	0.0169	0.2496	0.2577
λ	2.0000	2.0021	0.0253	1.9961	2.0081
μ	4.0000	4.0019	0.0509	3.9898	4.0139
ρ	0.5000	0.5003	0.0088	0.4983	0.5024

Table A.5. Verification of Parameters M/M/1 Queue with $\rho = 0.75$ and 95% Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 95%	CI Upper 95%
L	3.0000	3.0186	0.2300	2.9533	3.0840
L_q	2.2500	2.2679	0.2230	2.2045	2.3313
ω	1.0000	1.0058	0.0713	0.9856	1.0261
ω_q	0.7500	0.7555	0.0703	0.7355	0.7755
λ	3.0000	2.9990	0.0273	2.9912	3.0068
μ	4.0000	3.9950	0.0362	3.9847	4.0053
ρ	0.7500	0.7507	0.0096	0.7480	0.7535

Table A.6. Verification of Parameters M/M/1 Queue with $\rho = 0.75$ and 90% Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
L	3.0000	3.0186	0.2300	2.9641	3.0732
L_q	2.2500	2.2679	0.2230	2.2150	2.3208
ω	1.0000	1.0058	0.0713	0.9889	1.0227
ω_q	0.7500	0.7555	0.0703	0.7389	0.7722
λ	3.0000	2.9990	0.0273	2.9925	3.0055
μ	4.0000	3.9950	0.0362	3.9864	4.0036
ρ	0.7500	0.7507	0.0096	0.7484	0.7530

Table A.7. Verification of Parameters M/M/1 Queue with $\rho = 0.8$ and 95% Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 95%	CI Upper 95%
L	4.0000	4.0292	0.2024	3.9717	4.0867
L_q	3.2000	3.2279	0.1987	3.1715	3.2844
ω	0.5000	0.5033	0.0240	0.4964	0.5101
ω_q	0.4000	0.4031	0.0238	0.3964	0.4099
λ	8.0000	8.0041	0.0506	7.9897	8.0185
μ	10.0000	9.9893	0.0576	9.9729	10.0057
ρ	0.8000	0.8012	0.0058	0.7996	0.8029

Table A.8. Verification of Parameters M/M/1 Queue with $\rho = 0.8$ and 90% Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
L	4.0000	4.0292	0.2024	3.9812	4.0772
L_q	3.2000	3.2279	0.1987	3.1808	3.2751
ω	0.5000	0.5033	0.0240	0.4976	0.5089
ω_q	0.4000	0.4031	0.0238	0.3975	0.4088
λ	8.0000	8.0041	0.0506	7.9921	8.0161
μ	10.0000	9.9893	0.0576	9.9756	10.0029
ρ	0.8000	0.8012	0.0058	0.7999	0.8026

Table A.9. Verification of Parameters M/M/1 Queue with $\rho = 0.90$ and 95% Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 95%	CI Upper 95%
L	9.0000	9.0117	0.9549	8.7403	9.2831
L_q	8.1000	8.1121	0.9491	7.8423	8.3818
ω	1.0000	1.0016	0.1020	0.9726	1.0306
ω_q	0.9000	0.9016	0.1017	0.8727	0.9305
λ	9.0000	8.9923	0.0486	8.9784	9.0061
μ	10.0000	9.9962	0.0505	9.9818	10.0106
ρ	0.9000	0.8996	0.0073	0.8975	0.9017

Table A.10. Verification of Parameters M/M/1 Queue with $\rho = 0.90$ and 90% Confidence

Measure	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
L	9.0000	9.0117	0.9549	8.7853	9.2381
L_q	8.1000	8.1121	0.9491	7.8870	8.3371
ω	1.0000	1.0016	0.1020	0.9774	1.0258
ω_q	0.9000	0.9016	0.1017	0.8775	0.9257
λ	9.0000	8.9923	0.0486	8.9807	9.0038
μ	10.0000	9.9962	0.0505	9.9842	10.0082
ρ	0.9000	0.8996	0.0073	0.8979	0.9014

Table A.11. M/M/1/K Probability Values vs Simulation Values for $K = 10$ and $\rho = 1$ 95% Confidence

K	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 95 %	CI Upper 95 %
0	0.090909091	0.090467	0.004955	0.089059	0.091875
1	0.090909091	0.090657	0.004338	0.089425	0.091890
2	0.090909091	0.090548	0.002727	0.089773	0.091323
3	0.090909091	0.090741	0.002741	0.089963	0.091520
4	0.090909091	0.090538	0.002471	0.089836	0.091241
5	0.090909091	0.090763	0.002209	0.090135	0.091391
6	0.090909091	0.090762	0.002063	0.090176	0.091349
7	0.090909091	0.090684	0.002284	0.090035	0.091333
8	0.090909091	0.091527	0.002962	0.090686	0.092369
9	0.090909091	0.091418	0.003493	0.090425	0.092410
10	0.090909091	0.091893	0.004287	0.090675	0.093111

Table A.12. M/M/1/K Probability Values vs Simulation Values for $K = 10$ and $\rho = 1$
90% Confidence

K	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
0	0.090909091	0.090467	0.004955	0.089292	0.091642
1	0.090909091	0.090657	0.004338	0.089629	0.091686
2	0.090909091	0.090548	0.002727	0.089902	0.091195
3	0.090909091	0.090741	0.002741	0.090092	0.091391
4	0.090909091	0.090538	0.002471	0.089953	0.091124
5	0.090909091	0.090763	0.002209	0.090239	0.091287
6	0.090909091	0.090762	0.002063	0.090273	0.091252
7	0.090909091	0.090684	0.002284	0.090143	0.091226
8	0.090909091	0.091527	0.002962	0.090825	0.092230
9	0.090909091	0.091418	0.003493	0.090590	0.092246
10	0.090909091	0.091893	0.004287	0.090876	0.092909

Table A.13. M/M/1/K Probability Values vs Simulation Values for $K = 15$ and $\rho = 1$
95% Confidence

K	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 95%	CI Upper 95%
0	0.0625	0.062197	0.003497	0.061204	0.063191
1	0.0625	0.062139	0.003137	0.061248	0.063031
2	0.0625	0.062281	0.002637	0.061532	0.063031
3	0.0625	0.062057	0.002451	0.061361	0.062754
4	0.0625	0.062023	0.002248	0.061384	0.062662
5	0.0625	0.062239	0.002111	0.061639	0.062839
6	0.0625	0.062583	0.001712	0.062097	0.063070
7	0.0625	0.062503	0.001788	0.061995	0.063011
8	0.0625	0.062697	0.001876	0.062164	0.063230
9	0.0625	0.062824	0.001811	0.062310	0.063339
10	0.0625	0.062727	0.001745	0.062231	0.063223
11	0.0625	0.062565	0.001969	0.062005	0.063124
12	0.0625	0.062794	0.002425	0.062105	0.063483
13	0.0625	0.062700	0.002627	0.061954	0.063447
14	0.0625	0.062804	0.002704	0.062035	0.063572
15	0.0625	0.062865	0.003139	0.061973	0.063757

Table A.14. M/M/1/K Probability Values vs Simulation Values for $K = 15$ and $\rho = 1$
90% Confidence

K	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
0	0.0625	0.062197	0.003497	0.061368	0.063026
1	0.0625	0.062139	0.003137	0.061396	0.062883
2	0.0625	0.062281	0.002637	0.061656	0.062906
3	0.0625	0.062057	0.002451	0.061476	0.062638
4	0.0625	0.062023	0.002248	0.061490	0.062556
5	0.0625	0.062239	0.002111	0.061739	0.062740
6	0.0625	0.062583	0.001712	0.062177	0.062989
7	0.0625	0.062503	0.001788	0.062079	0.062927
8	0.0625	0.062697	0.001876	0.062252	0.063142
9	0.0625	0.062824	0.001811	0.062395	0.063254
10	0.0625	0.062727	0.001745	0.062313	0.063141
11	0.0625	0.062565	0.001969	0.062098	0.063032
12	0.0625	0.062794	0.002425	0.062219	0.063369
13	0.0625	0.062700	0.002627	0.062078	0.063323
14	0.0625	0.062804	0.002704	0.062162	0.063445
15	0.0625	0.062865	0.003139	0.062121	0.063609

Table A.15. M/M/1/K Probability Values vs Simulation Values for $K = 25$ and $\rho = 1$
95% Confidence

K	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 95%	CI Upper 95%
0	0.038462	0.038717	0.003735	0.037656	0.039779
1	0.038462	0.038715	0.003679	0.037670	0.039761
2	0.038462	0.038608	0.003763	0.037539	0.039677
3	0.038462	0.038617	0.003844	0.037525	0.039710
4	0.038462	0.038485	0.003894	0.037378	0.039591
5	0.038462	0.038419	0.003749	0.037354	0.039485
6	0.038462	0.038217	0.003602	0.037193	0.039240
7	0.038462	0.038295	0.003590	0.037274	0.039315
8	0.038462	0.038072	0.003475	0.037084	0.039059
9	0.038462	0.038116	0.003431	0.037141	0.039091
10	0.038462	0.038007	0.003445	0.037028	0.038986
11	0.038462	0.037949	0.003407	0.036981	0.038917
12	0.038462	0.037760	0.003403	0.036793	0.038727
13	0.038462	0.037709	0.003442	0.036731	0.038687
14	0.038462	0.037521	0.003465	0.036536	0.038506
15	0.038462	0.037691	0.003342	0.036741	0.038640
16	0.038462	0.037701	0.003262	0.036774	0.038628
17	0.038462	0.037841	0.003516	0.036842	0.038841
18	0.038462	0.037813	0.003551	0.036804	0.038822
19	0.038462	0.039919	0.012745	0.036297	0.043541
20	0.038462	0.032988	0.036515	0.022610	0.043366
21	0.038462	0.040582	0.017396	0.035638	0.045526
22	0.038462	0.040484	0.016939	0.035670	0.045298
23	0.038462	0.040560	0.017306	0.035642	0.045478
24	0.038462	0.041079	0.021248	0.035041	0.047118
25	0.038462	0.040136	0.013422	0.036321	0.043950

Table A.16. M/M/1/K Probability Values vs Simulation Values for $K = 25$ and $\rho = 1$
90% Confidence

K	Theoretical Values	Calc. Avg.	Calc. Std. Dev.	CI Lower 90%	CI Upper 90%
0	0.038462	0.038717	0.003735	0.037832	0.039603
1	0.038462	0.038715	0.003679	0.037843	0.039588
2	0.038462	0.038608	0.003763	0.037716	0.039500
3	0.038462	0.038617	0.003844	0.037706	0.039529
4	0.038462	0.038485	0.003894	0.037561	0.039408
5	0.038462	0.038419	0.003749	0.037530	0.039308
6	0.038462	0.038217	0.003602	0.037362	0.039071
7	0.038462	0.038295	0.003590	0.037443	0.039146
8	0.038462	0.038072	0.003475	0.037248	0.038896
9	0.038462	0.038116	0.003431	0.037302	0.038929
10	0.038462	0.038007	0.003445	0.037190	0.038824
11	0.038462	0.037949	0.003407	0.037141	0.038757
12	0.038462	0.037760	0.003403	0.036953	0.038567
13	0.038462	0.037709	0.003442	0.036893	0.038525
14	0.038462	0.037521	0.003465	0.036699	0.038342
15	0.038462	0.037691	0.003342	0.036898	0.038483
16	0.038462	0.037701	0.003262	0.036928	0.038475
17	0.038462	0.037841	0.003516	0.037008	0.038675
18	0.038462	0.037813	0.003551	0.036971	0.038655
19	0.038462	0.039919	0.012745	0.036897	0.042940
20	0.038462	0.032988	0.036515	0.024330	0.041646
21	0.038462	0.040582	0.017396	0.036457	0.044706
22	0.038462	0.040484	0.016939	0.036468	0.044501
23	0.038462	0.040560	0.017306	0.036457	0.044663
24	0.038462	0.041079	0.021248	0.036042	0.046117
25	0.038462	0.040136	0.013422	0.036954	0.043318