QUEUING SYSTEMS WITH PREFERRED SERVICE START TIMES AND MULTIPLE CUSTOMER CLASSES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MELİS BORAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
OPERATIONAL RESEARCH

SEPTEMBER 2020

Approval of the thesis:

QUEUING SYSTEMS WITH PREFERRED SERVICE START TIMES AND MULTIPLE CUSTOMER CLASSES

submitted by **MELIS BORAN** in partial fulfillment of the requirements for the degree of **Master of Science in Operational Research Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Sinan Gürel
Head of Department, **Operational Research** _____

Assist. Prof. Dr. Sakine Batun
Supervisor, **Industrial Engineering, METU** _____

Assist. Prof. Dr. Bahar Çavdar
Co-supervisor, **ETID, Texas A&M University** _____

Examining Committee Members:

Prof. Dr. Yasemin Serin
Industrial Engineering, METU _____

Assist. Prof. Dr. Sakine Batun
Industrial Engineering, METU _____

Assist. Prof. Dr. Bahar Çavdar
ETID, Texas A&M University _____

Assoc. Prof. Dr. Seçil Savaşaneril
Industrial Engineering, METU _____

Assist. Prof. Dr. Tuğçe Işık
Industrial Engineering, Clemson University _____

Date: 15.09.2020

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Melis Boran

Signature :

ABSTRACT

QUEUING SYSTEMS WITH PREFERRED SERVICE START TIMES AND MULTIPLE CUSTOMER CLASSES

Boran, Melis
M.S., Department of Operational Research
Supervisor: Assist. Prof. Dr. Sakine Batun
Co-Supervisor: Assist. Prof. Dr. Bahar Çavdar

September 2020, 79 pages

Motivated by the operational problems in curbside pickup systems, in this thesis, a capacity allocation problem in a queuing system, where arriving customers have preferred service delivery times, is studied. In the illustrated system, customers may have different priorities. All requests are assumed to be served before or during their requested time period. In order to increase the generality of the model, actions of early service, rejecting a customer request, and outsourcing (or equivalently working overtime) are allowed for the service provider. We model this problem as a Markov Decision Process. We analyze the structural properties to identify a set of suboptimal solutions, and use these results to develop efficient solution methods. We develop state space aggregation methods to solve problem instances of larger size.

Keywords: Capacity allocation, Markov Decision Process, Customer preferences, Curbside pickup systems

ÖZ

TERCİH EDİLEN HİZMET BAŞLANGIÇ ZAMANLI VE ÇOKLU MÜŞTERİ SINIFLI KUYRUK SİSTEMLERİ

Boran, Melis
Yüksek Lisans, Yöneylem Araştırması Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Sakine Batun
Ortak Tez Yöneticisi: Dr. Öğr. Üyesi. Bahar Çavdar

Eylül 2020 , 79 sayfa

Bu tezde arabayla teslim al sistemlerinde ortaya çıkan operasyonel problemlerden yola çıkılarak, gelen müşterilerin servis teslim sürelerini tercih ettikleri bir kuyruk sisteminde kapasite tahsisi problemi incelenmiştir. İncelenen sistemde müşteriler farklı önceliklere sahip olabilirler. Tüm taleplerin talep edilen zamandan önce veya talep edilen zamanda karşılandığı varsayılır. Modelin genelliğini artırmak için, erken servis, bir müşteri talebini reddetme ve servis sağlayıcı için dış kaynak kullanımı (alternatif olarak fazla mesai) gibi eylemlere izin verilir. Ele alınan problem bir Markov Karar Süreci olarak modellenmiştir. En iyi hizmet politikasını oluşturmak için bazı yöntemler kullanılmıştır; ilk olarak, uygun olmayan çözümler kümesini karakterize etmek için yapısal sonuçlar önerilmiştir ve bu sonuçlar verimli hesaplamalar için çözüm yöntemleriyle entegre kullanılmıştır. Ayrıca, daha büyük boyutlardaki problemleri çözebilmek için bazı probleme özgü yaklaşım ve durum uzayı toplama yöntemleri geliştirilmiştir.

Anahtar Kelimeler: Kapasite tahsisi, Markov Karar Süreci, Müşteri tercihleri, Arabayla teslim al sistemleri

To my sister, Beril, who suffered through it with me

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my supervisor and my co-supervisor, for their continuous support, guidance, encouragement, and trust. I have learned a lot from them, and it was a great privilege to work and study under their guidance.

I am also very grateful to Assist. Prof. Dr. Bahar Çavdar and Assist. Prof. Dr. Tuğçe Işık for their valuable insights and essential contributions during this study. It was a great opportunity for me to study with them.

I would like to convey my thanks to my jury members Prof. Dr. Yasemin Serin, Assoc. Prof. Dr. ̇Seçil Savaşaneril, and Assist. Prof. Dr. Sakine Batun for allocating their time to my thesis and for their precious comments in enhancement of my thesis.

I want to share my deepest gratitude with my family, my father H. İbrahim Boran, my mother Tülay Boran, my brother Selim Boran and my sisters Merve Boran, Beril Boran. I owe them my never-ending eagerness to improve myself, my curiosity, and my ambition. I am thankful for their love and guidance in whatever I pursue.

Last but not least, I would like to thank everyone who did not lose confidence in me throughout my study and who touched my soul.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF ABBREVIATIONS

ABBREVIATIONS

MDP	Markov Decision Process
LP	Linear Program
DLP	Dual Program
BLP	Basic Linear Program
RLP	Reduced Linear Program
MP	Myopic Policy
AP	Always-serve Policy
AG	Absolute Gap
MAP	Matched Action Percentage

CHAPTER 1

INTRODUCTION

Advancements in online platforms and changes in customer behavior have been transforming the market operations and strategies in various industries. In recent years, traditional retail market has been hit hard by the availability of online shopping as digitization redefines the meaning of "go shopping." Nielsen [1], reports that about one-fourth of global consumers shop for groceries online, and 55% say they would buy groceries online. Further, the Millennials and Generation Z, the future drivers of the economy, shop for about 60% of their purchases online [2]. So, noticing the upcoming online channels' opportunities, retail agents increasingly adopt multi-channel, e-commerce models, and they become a serious threat to traditional brick-and-mortar retailers. To maintain their competitiveness, the traditional retail marketers should adapt their systems, enhance the shopping experience that they provide and meet consumers' evolving habits [3].

One strategy that has been attracting attention is establishing "click and collect" services, which integrates online and offline commerce. These services allow customers to order and purchase online for pickup at a store or another location. Even though currently just over 10% of global consumers prefer these services, more than half are willing to use these online options (e.g., in-store, drive-through, and curbside pickup) in the future [1]. Among these options, curbside pickup allows customers to pick up their orders from the store without walking into the building and without even having to leave their car and has already shown its efficiency despite being relatively new compared to other similar mechanisms [2].

Consumers favor retailers which provide efficient and fast service solutions. Thanks to curbside pickup, retailers provide the convenience, ease, and flexibility of online

ordering without the wait for delivery. It eliminates in-store navigation, standing in line for checkout, and the problem of finding items in the store [4]. Moreover, retailers have the opportunity to satisfy their customers quickly by using their existing resources, enhance their brand loyalty by providing an excellent customer experience and save the cost of delivery charges. This benefits both parties, even though it keeps customers away from potential impulse buys and upselling opportunities.

US Retailers like Walmart, Kroger, and Target are early pioneers of this system. Powered by a fast expansion from these firms, grocery curbside pickup is on target to become a $35 billion industry by next year. Today with more than 2000 curbside pickup locations, Walmart leads the market, and it is estimated that it drives $7.4 billion in revenue and accounts for 33% of total digital sales [4]. Kroger, Target, and Whole Foods are in the race to add the curbside pickup service to their charters. Target currently has its curbside pickup option available in around 1000 stores with a small grocery selection. To its credit, Walmart has begun to expand curbside for select items in health and beauty, household essentials, and party and craft supplies. Kroger has more curbside locations than Target, around 1600, but charges a $4.99 fee [4].

In curbside pickup systems, each customer request is received with a preferred service completion time; and there can be a time gap between when the service is requested and when it needs to be fulfilled. Unlike traditional service systems, it may not be appropriate to serve requests in the order they are received or as soon as possible. Certainly, compared to traditional delivery systems, this service scheme provides more flexibility to consumers while revealing new service challenges to retailers, who have more control over the service process. For example, for a grocery pick up, starting the service close to the desired pick up time can be preferable to maintain the products' freshness. While the company cares about customer satisfaction, it also needs to consider any capacity restrictions, since the demand over the next time slots is unpredictable. Thus, the planning and scheduling of order picking/preparation operations are critical for the success of these new services.

Although we mainly consider the retail industry, there are similar considerations in other service sectors. For instance, companies operating in delivery, maintenance, and even in healthcare, may allow customers to select time windows rather than specific

appointment times. These time constraints taking customer preferences into account result in a higher level of complexity in operations planning. Moreover, although service providers might not able to serve a job in its entirety before the scheduled completion time, there can be parts of service that can be completed in advance.

Our study is mainly motivated by the above challenges that rise in curbside pickup systems. The problem is briefly introduced in this chapter, and the detailed definition and formulation are provided in Chapter 3. Inspired from the work [5], this thesis considers a queuing system with a fixed length customer order horizon where customers have preferred service completion times and different priorities to model this new retailing strategy. We define two classes of customers, high-priority and low-priority. High-priority customers are loyal and/or members of our target group, so the fulfillment of their requests is promised. However, requests of low-priority customers can be rejected upon arrival.

We model the described system as a discrete-time Markov Decision Process (MDP). It is assumed that the service requests are received and served in discrete time periods. In each time period t, a number of new service requests with preferred service completion times are observed according to a stochastic arrival process. Customers' preferred service completion times can be anywhere in the customer order horizon, the next K periods, including the current period t.

The service capacity is constant over all time periods and at most M jobs can be served in any given period. The server provider simultaneously decides whom to reject among new arrivals and whom to serve in each time period. A request that is rejected incurs rejection cost. An admitted request can be fulfilled either on time or early. Note that the service provider incurs early service costs in case of early service. All admitted job requests must be served by their preferred service completion time, late services are not allowed and customers do not abandon the queue. In each period, all remaining unserved requests in the system are placed in a queue to be served later, until their service completion time.

We address admission control and the dynamic capacity allocation decisions jointly. Our objective is to find an admission and capacity allocation policy that results in the minimum long-run average cost over an infinite planning horizon, that requires an

explicit consideration of the trade-off between immediate and expected future benefits when choosing the current action.

The size of the formulated MDP rapidly increases with the number of time periods in the customer order horizon, and the maximum number of arrivals that can be observed in each period. Finding the optimal policy is computationally expensive, especially using standard methods. To address this computational difficulty, we analyze the structural characteristics of the suboptimal solutions, and integrate these characterization into generic solution methods (e.g., linear programming) for efficient computations. We also consider the exact and approximate state space reduction methods to be able to solve larger sizes of instances.

To our knowledge, this thesis is the first study analyzing the structure and the properties of the optimal policy in queuing systems where customers have preferred service completion times and dynamic priorities. Our contributions are as follows:

- We provide an MDP formulation for the joint admission control and capacity allocation problem in queues with constant capacity, earliness, and rejection costs.
- Our model uses information available on the arrival process to inform capacity management decisions throughout the rolling planning horizon.
- We provide structural results on the optimal admission and allocation policies. These results are easy to apply since they are based on system parameters and independent of the random arrival process. Thus, they can be easily integrated with different solution approaches.
- We develop problem-specific aggregation methods, that find approximate-optimal policies, and provide a sensitivity analysis on both the resulting optimality gap and computational performance.

The rest of the thesis is organized as follows; in the next chapter, we present an overview of the existing literature in Markov Decision Processes (MDP) and their applications. In Chapter 3, we define the problem, state our assumptions, and formulate our MDP model. We present our solution methodology and structural results in

Chapter 4. Chapter 5 presents the computational results evaluating the performance of the proposed solution approaches. The thesis is concluded in Chapter 6 with a brief conclusion and discussion on our key results, as well as our future research directions.

CHAPTER 2

LITERATURE REVIEW

The problems of admission control and dynamic resource allocation in queuing systems have been considered in a variety of contexts, and they are widely studied in the literature. However, most of the papers considering these problems discuss them in isolation. Both problems are of interest when the resources are scarce. In admission control problems, the objective is to strategically admit or reject jobs (or service requests) based on the availability of resources, whereas, in allocation problems, researchers investigate the effective use of resources in the face of demand uncertainty and associated costs. Markov decision processes have many applications for such problems. In a similar vein, our work considers admission control and the allocation of resources when the demand is random, the capacity is scarce, and shortages are costly. However, our focus is unique since we consider that requests may arrive in the system earlier than their preferred service completion time.

Our work is related to several research areas. We first review the existing literature on Markov decision processes, which has a dominant role in shaping our solution methodology. Then, we review relevant papers on service systems, such as manufacturing and health-care, to illustrate the validity of the considered problem in real-life applications.

2.1 Markov decision processes: concepts and approaches

We first introduce the necessary concepts from Markov Decision Processes. In the most general sense, an MDP is a continuous or discrete-time stochastic control process. In this study, we only consider the discrete-time formulation. We are given a

set of states $\mathcal{S}$. Suppose $\mathcal{S}$ is finite. At each time period $t = 0, 1, ...$, the process is in some state $s_t \in S$, and the controller choose an action d from the finite set D_{s_t} of allowable actions in s_t. Following this decision, we immediately incur a cost $u(s_t, d)$ and the process responds at the next time $t + 1$ by randomly moving into a state s_{t+1} with probability $p(s_{t+1}|s_t, d)$. The state transitions certainly possess the Markov property, the probability that the process enters state s_{t+1} is determined only by the state s_t and the chosen action d and is independent of all previous decisions.

The main goal of an MDP is to identify an optimal policy for the controller, which minimizes or maximizes a given objective function of some cumulative function of costs or rewards. There are two approaches to solve this type of optimization problem. First, the problem can be expressed as a linear program and thus solved with some generic algorithms. Secondly, there are iterative techniques, such as policy iteration and value iteration, that are known to converge fast on the optimal policy [6]. Although the linear program is a very robust approach, neither value iteration nor policy iteration makes significant use of the structural features of the underlying dynamic model [7]. That is in the linear programming approach it is easier to impose structural conditions of the dynamic model on the linear program. This fact is one of the main motivations for using the linear programming approach in this study.

The computational complexity of MDPs has been an emerged notion. From the theoretical point of view, since all MDPs can be formulated as a linear program, all can be solved in a number of arithmetic operations polynomial in the numbers of states and actions and the bit-size of the input data [6]. Additionally, Papadimitriou and Tsitsiklis [6] showed that MDPs with the average cost optimality criterion can be solved in strongly polynomial-time so that a polynomial in the numbers of states and actions bounds the number of arithmetic operations needed to solve.

The fact that the linear programming formulation of MDPs can be solved in polynomial time is not particularly comforting. From the applications point of view, solving an MDP optimally becomes a particularly relevant challenge. For instance, the application of Markov decision theory to the control of queueing systems often leads to models with enormous state and action spaces. Hence, the computation of optimal policies is expensive or almost impossible for most practical models [8]. Fortunately,

however, there usually is a lot of structural properties that can be exploited in such problems. This process usually involves showing that the optimal value function has certain convexity, concavity, or multimodularity properties. For the same purpose, Koole [9] put forward a systematic approach for deriving monotonicity results of optimal policies for various queueing and resource sharing models. In his ongoing work [10], he provided more generalized formulations.

There are indeed two main approaches to MDPs: first, the computational performance requirements are relaxed, then traditional techniques are used, and, second researchers focus on narrower classes of MDPs that have exploitable structure. However, in the literature, there are some attempts to prevent computational sacrifices. Kallenberg [11] provided some classification algorithms, then Feinberg and Yang [12] showed the complexity of such algorithms. They also showed that these algorithms are useful tools not only for the classification problem but also for the state-space decomposition procedure. Another way to overcome computational challenges is the use of action elimination procedures that work well with value iteration. Lasserre [13] yielded sufficient conditions that permit the detection of optimal and non-optimal actions. However, none of these methods permanently eliminates non-optimal actions from the actions-space; all need internal tests in various stages of the solution algorithm.

The solution methodology presented in this thesis differs from the previous approaches; first, we propose an action elimination procedure based on problem-specific properties, and secondly, we work on the linear program formulation with a reduced state space. We state some simple conditions, in terms of problem parameters, that easily identify suboptimal actions. The actions found to be suboptimal are removed from the action space and, and a linear program is reformulated on that reduced action space.

2.2 Applications in service systems

There is a vast literature on service capacity management. Capacity allocation studies consider operational tools including admission, priority, and dynamic resource allocation or combinations of them. We give close attention to such health-care and man-

ufacturing problems, since time-windows and priorities are commonly considered in these domains.

Patient scheduling problem consists of simultaneous admission and scheduling of patients. Models developed to study these problems often consider preliminary priority classes associated with patients' needs. Studies of this area consider distinct healthcare systems.

Green et al. [14] focus on hospital diagnostic facilities, such as magnetic resonance imaging (MRI) centers, providing service for different patient groups: outpatients, inpatients and emergency patients. Emergency patients belong to a preemptive class and whenever an emergency patient is admitted to the system, he must be served. Their analysis shapes around two interrelated tasks: designing the outpatient appointment schedule and dynamic priority rules for admitting patient into service. They formulate the problem as a discrete-time MDP and propose monotone "switching curve" policies to solve . A similar problem is also addressed by Patrick et al. [15]. They especially consider resource management in the form of dynamic allocation of available capacity to incoming demand to achieve wait-time targets in a cost-effective manner. They model the scheduling process as MDP. They solve the equivalent linear program through approximate dynamic programming.

Ayvaz and Huh [16] widen the problem considerations, and work on the joint of admission control and dynamic allocation of hospital capacity, among several customer types. They adopt a dynamic-programming approach and show the monotonicity properties of the optimal policy. Because of difficulties on finding the optimal policy, they propose a simple threshold heuristic policy that performs well in experiments.

Primary care offices have also been under consideration, since these systems suffers from last-minute cancellations and no-show behavior of outpatients. Green and Savin [17] treat the problem as a single-queueing system. Their results demonstrate the usefulness of the queueing models in providing guidance on identifying patient panel sizes for medical practices. The three more recent studies [18], [19], and [20] additionally take patients preferences into consideration. They all propose well-performing heuristic procedures.

We also reviewed works that considered capacity allocation and scheduling for manufacturing of a single product or multiple products for multiple customer classes under uncertain arrivals. In the literature, the idea that it might be preferable to postpone jobs until a certain time is often modeled using time windows. There are several publications that consider scheduling of jobs with time windows. Slotnick [21] and Ouelhadj and Petrovic [22] provide an overview of this research area and represent commonly used approaches.

Duenyas [23] formulates the problem of sequencing a single product in a manufacturing system with multiple classes of customers as a semi-Markov decision process and examines the due date setting policy. Following this idea, Carr and Duenyas [24] address a combined problem of admission control and scheduling in a production system with two product classes. The first class of products is made-to-stock, and the producer has obligations about their production. However, the second class of products is made-to-order, and the firm has the option of admission or rejection of a particular order. In this sense this study is similar to ours. They use a single two-class M/M/1 queue and propose a switching curve policy. Mönch et al. [25] study the single machine scheduling problem minimizing earliness-tardiness with an upper bound on allowable total tardiness characterized by a maximum allowable time value. They suggest several two-phased heuristic procedures, which are based on genetic algorithms and dominance properties.

Zhao and Lian [26] study an inventory system. This system consists of two demand classes with different priorities, as in our case. In contrast to our model, unsatisfied demand in the high-priority class are lost, whereas those in the low-priority class are backlogged, following different cost schedules. They formulate the problem as a dynamic programming model and a construct optimal replenishment policy.

This thesis is inspired by the work of Çavdar and Işık [5]. In [5], the system controller systematically makes the early service decision. They show that, for systems with short customer order horizons, a class of threshold policies are optimal for the capacity allocation. For general systems, they develop heuristic policies based on threshold structures. In this thesis, we consider a more general problem with multiple customer types and rejection of job requests.

CHAPTER 3

PROBLEM DEFINITION AND FORMULATION

In this chapter, we provide a detailed problem definition and the formulation. We study a queuing system with constant server capacity M and a fixed customer order horizon. In our problem setting, customers arrive at each discrete time period with a service request for the current or a future period. That is, a customer arriving at time t makes a request to be served in period $t + j$, where $0 \leq j < K$. Here K represent the length of the customer order horizon. We consider two types of customers, high-priority and low-priority. We use index $i = 1$ for the former and $i = 2$ for the latter. High-priority customer requests must be accepted to the queue. However, the controller can reject some low-priority requests at the time of job request. Note that once a low-priority job is accepted to the queue, it has to be served. Considering the practical implications of joining the queue before the requested service time, we assume that low-priority customers who wait in the queue at least one period are considered to be high-priority. That is, their priority level is shifted from low to high, and they are treated as high-priority customers thereafter. Independent of the customer's priority class, customers do not abandon the system.

Our main goal is to allocate the available capacity to the admitted service requests waiting in the queue to maximize the long-run average profit of this system. Every service completion provides a constant revenue to the system. By interpreting the rejection cost as the cost of low-priority customers' dissatisfaction (hence customers do not leave the queue and are ultimately served), it can also be assumed that jobs are not lost. As a result, we can consider the system as a pure waiting system, and we can conclude that the collected reward is independent of the taken allocation decision.

Therefore, our focus is on identifying the joint admission and capacity allocation policy for the service provider to minimize operation costs in the long-run.

We model this system under the following set of simplifying assumptions, as a discrete-time infinite-horizon Markov decision process (MDP). The service duration of each job is assumed to be exactly one period and the decision epochs are just after customer arrivals at each discrete time period, which means the process is observed by the controller at these discrete time points when the arrivals are realized. By these assumptions, it is certain that at the beginning of each time period all M servers are available. Therefore, we do not need to keep track of server availability in the system state. Job requests arrive for service are first placed in another queue, "arriving job queue." At each decision epoch, the controller observes the number of jobs currently in the queue waiting to receive service. On this basis, he simultaneously decides how many low-priority jobs to admit from the arriving job queue, and how many job requests, either high-priority or low-priority, to serve. Admitted jobs eventually receive service in the current or a future period; low-priority jobs that are not admitted never enter the system. Therefore, it is compulsory to track the arriving customer request and the waiting customer requests at time period t to determine the state of the system. Let $x^t = x$ denote the load of the queue at time t, immediately prior to decision, and $x = \{x_{i,j} \mid i = 1, 2 \text{ and } 0 \leq j < K\}$, where $x_{i,j}$ is the number of i priority jobs in the queue that are waiting to be served j periods later. Note that these are the jobs that are already accepted to the queue and have to be served. Let $a^t = a = \{a_{i,j} \mid i = 1, 2 \text{ and } 0 \leq j < K\}$ represent the arrivals at time t, where $a_{i,j}$ is the number of priority class i arrivals that are willing to be served j periods later. We represent the system state s^t by the pair (x^t, a^t), and we denote the set of all possible states by $\mathcal{S}$.

It is assumed that customer arrivals follow a known discrete distribution with arrival rate λ. The probability of observing a many arrivals of i-priority requesting service for $t + j$ is computed respect to general arrival distribution with arrival rate $\lambda_{i,j}$, $i = 1, 2$ and $0 \leq j < K$. The arrivals of each priority class and for each time period are independent of the number of jobs waiting to be served, and as well independent for all i and j. Notations for the arrival process are as follows:

- λ: general arrival rate,
- q_i: probability that a customer arriving at time t belongs to priority class i,
- v_j: probability that a customer arriving at time t requests a service for $t + j$,
- $\lambda_{i,j}$: arrival rate of customers of class i who request a service for $t + j$,

$$\lambda_{i,j} = q_i v_j \lambda \text{ for } i = 1, 2 \text{ and } 0 \leq j < K \tag{3.1}$$

- $p_{i,j}(a)$: probability of observing a customers of class i who request a service for $t + j$.

Notice that when the customer arrivals are modeled by truncated distributions, both, the state space $\mathcal{S}$ and the action space $\mathcal{D}$ are finite. Therefore to avoid the curse of dimensionality due to large state and action spaces we work with truncated arrival distribution, so that in each period, at most A arriving job requests that are required to be served in $t + j$ can be observed for all i.

Arrivals are followed by decision epochs. At each decision epoch, a controller simultaneously decides which job requests to reject among low-priority arrivals and which job requests to serve, considering the server capacity. A low-priority job request can be rejected, with an additional cost c_r. Once arriving job request are admitted to the queue, each job waiting in the queue has to be served by its requested service time. That means, while early service is allowed, late service is not. If the number of jobs that has the current period as their preferred service completion time exceeds the available capacity, we assume that these jobs can be served by working in overtime or outsourcing at an additional cost c_o per job. This is also equivalent to not serving a job by its requested service time, and paying a penalty of c_o to the customer. In case of early service, the server incurs early service cost per period per job. This cost item represents holding/handling cost of the order until customer pickup or customer dissatisfaction due to receiving the service prior to the requested time. In case of latter, it is natural to expect the early service cost to be higher for high-type customers. Therefore, for generality, we distinguish the early service cost by priority class; c_i^e is the early service cost per period for priority class i. For example, if a high-priority customer's request is served 4 periods in advance, we incur $4c_1^e$ as early service cost.

The controller's decision varies from strategically idling the servers in order to avoid overtime costs to taking the least costly action in each state. Note that all admitted jobs for the current period must be served even if the capacity is not enough. Therefore, early service is suboptimal unless there is remaining capacity after serving job requests for the current period, because the system would be incurred both early service and overtime costs unnecessarily. Similarly, for system with shorter customer horizons, if rejection occurs, early service is suboptimal for low-priority jobs because we would be incurring both rejection and early service costs unnecessarily. We denote an action by $d = (r, y)$ where $r = \{r_j \mid 0 \leq j < K\}$ itself represents the number rejected low-priority job requests that are for j periods later, and $y = \{y_{i,j} \mid i = 1, 2 \text{ and } 0 \leq j < K\}$ with $y_{i,j}$ is the number of priority class i jobs requested to be served at $t + j$ but served at time t. Notice that, $r_j \leq a_{2,j}$ holds for all j, since only arriving request can be rejected. Similarly, we have $y_{1,j} \leq x_{1,j} + a_{1,j}$ and $y_{2,j} \leq x_{2,j} + a_{2,j} - r_j$ for all j. Therefore, the action space is limited by the state space $\mathcal{S}$. We denote the set of all possible actions by $\mathcal{D}$, and since not all actions are allowed in each state we use $\mathcal{D}_s$ to denote the set of allowable actions in state s.

Given a current state s^t an action d, and a realization of arrivals a^{t+1} the next state can be given by the following transition function $f(.)$ and the associated transition probability $P(s^{t+1} \mid s^t, d\,)$.

$$s^{t+1} = f(s^t,\ d,\ a^{t+1}) = (x^{t+1}, a^{t+1}) \tag{3.2}$$

where

$$x^{t+1} = \begin{cases} x_{1,0}^{t+1} = ((x_{1,1}^t + a_{1,1}^t) - y_{1,1^t}\,) + ((x_{2,1^t} + (a_{2,1^t} - r_1^t\,)) - y_{2,1}^t\,) \\ x_{1,j}^{t+1} = (x_{1,j+1}^t + a_{1,j+1}^t) - y_{1,j+1}^t), \text{ for } 1 \leq j < K \\ x_{2,0} = 0 \\ x_{2,j}^{t+1} = (x_{2,j+1}^t + (a_{2,j+1}^t - r_{j+1}^t\,)) - y_{2,j+1}^t), \text{ for } 1 \leq j < K \end{cases}$$

As the transition function indicates, whenever an action is taken, the remaining unserved job requests in the system are placed in the queue to be served later. Then, the associated transition probabilities are computed as follows:

$$P(s^{t+1}\,|s^t, d\,) = \begin{cases} \prod\limits_{\substack{i=1,2 \\ j\in[1,K-1]}} p_{i,j}(a_{i,j}^{t+1}\,), \text{ if } w \\ 0, \text{ if not } w \end{cases} \tag{3.3}$$

where w is the event that

$$x^{t+1} = \begin{cases} x_{1,0}^{t+1} = ((x_{1,1}^t + a_{1,1}^t) - y_{1,1^t}\,) + ((x_{2,1^t} + (a_{2,1^t} - r_1^t\,)) - y_{2,1}^t\,) \\ x_{1,j}^{t+1} = (x_{1,j+1}^t + a_{1,j+1}^t) - y_{1,j+1}^t), \text{ for } 1 \le j < K \\ x_{2,0} = 0 \\ x_{2,j}^{t+1} = (x_{2,j+1}^t + (a_{2,j+1}^t - r_{j+1}^t\,)) - y_{2,j+1}^t), \text{ for } 1 \le j < K \end{cases}$$

The following example can provide a clearer understanding of the transition between the states and the probabilities.

Example: Consider a state when $K = 3$ at time t as follows:

$$s^t = (x^t, a^t) = [(1, 2, 0, 0, 2, 0), (1, 0, 0, 2, 1, 1)]$$

In this state, currently there is one high-priority customer who wants to be served now in the queue, such that $x_{1,0} = 1$. There are two high-priority, $x_{1,1} = 2$ and two low-priority, $x_{2,1} = 2$ job requests for $t + 1$, respectively. Additionally, there is one arriving high-priority job request to be fulfilled at time t, $a_{1,0} = 1$. There are arriving low-priority job requests as well: two for time t, one for time $t + 1$, and one for time $t + 2$; $a_{2,0} = 2$, $a_{2,1} = 1$, $a_{2,2} = 1$. Assume the controller's decision $d = (r, y) = [(1, 0, 0), (2, 0, 0, 1, 0, 0)]$. In the next period, the queue, x^{t+1}, looks like $(5, 0, 0, 0, 1, 0)$ before the realization of arrivals at $t+1$. So, the probability of reaching state $s^{t+1} = [(5, 0, 0, 0, 1, 0), (0, 0, 1, 0, 2, 0)]$ is the probability of observing arrivals of one high-priority job request due time $t + 2$ and two low-priority job requests due $t + 1$. Therefore, $P(s^{t+1}|s^t, d) = P(a^{t+1} = (0, 0, 1, 0, 2, 0)) = P(a_{10} = 2)P(a_{11} = 0)P(a_{12} = 0)P(a_{20} = 1)P(a_{21} = 0)P(a_{22} = 0)$. On the other hand, the probability

of reaching the state $[(5,0,0,0,0,0),(0,0,1,0,2,0)]$ is zero since post-decision states x^{t+1}'s do not match, $(5,0,0,0,1,0) \neq (5,0,0,0,0,0)$.

The total cost that is incurred in each period is the sum of the rejection, overtime and the early service costs. For a given period, the immediate cost we incur from the selection of d at state s^t is denoted by $c(s^t,d)$ and is computed as follows:

$$c(s^t,d) = c_o[M - \sum_{i=1}^{2} y_{i,0}]^+ + c_r \sum_{j=1}^{K-1} r_j + c_1^e \sum_{j=1}^{K-1} j y_{1,j} + c_2^e \sum_{j=1}^{K-1} j y_{2,j} \tag{3.4}$$

where $[\cdot]^+$ is the positive part function. The positive part of f is defined for all $x \in I$ by $[f(x)]^+ = max\{f(x),0\}$.

The proposed model aims to minimize the long-run average cost. The average reward optimality criterion ultimately depends on the limiting structure of the underlying stochastic process. That is why, at first, we should classify the model. A simple analysis concludes that this MDP is unichain. Regardless of the policy and the initial state, we can reach the state where there are no customers in the queue with some positive probability, due to the positive probability of observing zero customer arrival and a long enough inter-arrival time. So, the state with zero customers is recurrent. Since recurrence is a class property, all communicating states accessible from zero customer state make a recurrent class, and all remaining states form a set of transients.

For unichain models all stationary policies provide constant gain [27], a constant cost in our case, and the optimal gain is also known to be constant. Hence, a single optimality equation is enough to characterize optimal policies and their average costs. We have

$$0 = \min_{d \in \mathcal{D}} \{c\,(s,d) - g + \sum_{s' \in \mathcal{S}} P\{s'|s,d\} h(s'\,) - h(s)\}, \forall s \in \mathcal{S} \tag{3.5}$$

where g denotes the long-run average cost and $h(\cdot)$ is the bias, satisfying $\lim_{n\to\infty} \frac{E_n^\pi h(s_n)}{n} = 0$. As a result of $\mathcal{S}$ and $\mathcal{D}$ being finite, it follows that there exist actions that minimize the value function, thus the optimality equation surely attains a minimum value, since the existence of a deterministic stationary optimal policy is guaranteed in [27].

The dimensionality of the resulting MDP rapidly increases with the number of time periods in the customer order horizon K and the maximum number of arrivals A that can be observed in those periods for each customer class. Solving MDPs with a large and multidimensional state space and a large action space is computationally expensive, especially if one is using the standard iterative methods, Policy Iteration and Value Iteration. In this thesis, we adopt linear programming approach, as the basic solution tool, since it provides long-run average cost optimal policies with minimum effort once the associated model generated correctly. To increase computational performance, our first efforts are in revealing some structural results and characterizing suboptimal policies. To solve larger instances, we also address different state space reduction methods.

CHAPTER 4

SOLUTION METHODOLOGY

MDPs are solvable by Linear programming (LP) and some iterative techniques, such as Policy Iteration and Value Iteration. However, solving MDPs with large and multi-dimensional state and action spaces is computationally expensive. Thanks to its elegant theory, the LP formalization can facilitate computation. Furthermore, it allows the inclusion of new constraints easily and simplifies the sensitivity analysis.

Infinite horizon average cost problems are solvable in polynomial time, bounded by the number of states and actions, by the LP approach. Therefore, as we study the LP formulation of the considered MDP, our efforts are in enhancing computational performance by reducing the number of actions and states that needs to be included in our formulation.

In this chapter, we present several methods for solving the joint admission and capacity allocation problem we study. We first provide a LP model of the MDP. Since the size of the state space increases aggressively with minor increases in problem parameters, our computational power fails to satisfy the memory requirement of the LP formulation and its solution algorithm even for moderately-sized real life problems. Therefore, in order to decrease the computational requirements of the LP, associated with the size of MDP, we propose some action and state space reduction methods. First, to reduce the size of the action space, we partially characterize the optimal actions and successfully restrict the policy space, hence the feasible region of the corresponding LP. The presented structural results ensure a significant reduction in action space and strengthen our solution approach in terms of solution time and memory requirement. However, the LP model can not avoid from the curse of dimensionality, to overcome we secondly work on state space reduction methods. We achieve this

through exact and approximate approaches. We first distinguish recurrent states by a state space decomposition algorithm and reveal a reduced LP. Finally, we propose state aggregation methods and then aggregate MDP models. We also integrate those state space reduction techniques, with the proposed structural results, and observe their effects in solution quality and solution performance.

4.1 Linear Programming Formulation

Consider a general unichain MDP, described with tuple $< \mathcal{S}, \mathcal{D}, P, c >$ where $\mathcal{S}$, $\mathcal{D}$, P and c stand for state space, action space, transition probabilities and cost-to-go, respectively, under the long-run average cost optimality criterion. Let $J(\pi)$ be the long-run average cost when using a policy $\pi \in \Pi$,

$$J(\pi) = \lim_{n\to\infty} \frac{J_n(\pi)}{n} \tag{4.1}$$

where

$$J_n(\pi) = E^{\pi} \sum_{t=0}^{n-1} c(s^t, d) \tag{4.2}$$

and we define the minimum long-run average cost as

$$g^* = \min_{\pi\in\Pi} J(\pi) = J(\pi^*) \tag{4.3}$$

We are now concerned with the problem of finding a policy π^* minimizing the long-run average cost. Recalling the unichain optimality equation (3.5) we can relate the minimum policy problem to a linear program.

Theorem 1. *[28] Let (g, h) be a pair consisting a real number g and a measurable function $h : \mathcal{S} \to \mathbb{R}$ satisfying*

$$g + h(s) \leq\ c\,(s,d) + \sum_{s'\in\mathcal{S}} P\{s'|s,d\}h(s'\,) - h(s), \forall (s,d), s \in \mathcal{S}, d \in \mathcal{D}_s \tag{4.4}$$

and

$$\lim_{n\to\infty} E^{\pi} h(s_n)/n = 0, \forall \pi \in \Pi \tag{4.5}$$

Then

$$g \leq g^* \leq J(\pi^*) \tag{4.6}$$

A detailed proof of the above Theorem 1 is given in [28], as a consequence of above whenever there exists a pair $(g, h) \in \mathbb{R} \times V$, where V denotes the space of bounded real-valued functions on $\mathcal{S}$, satisfying (4.4), g^* is the largest g. This suggests the following LP model, with decision variables are g and $h(\cdot)$, expressing the average cost for the system in steady state and the bias, respectively.

$$\max \quad g \tag{4.7a}$$

$$\text{s.t.} \quad g + h(s) - \sum_{s' \in \mathcal{S}} P\{s'|s,d\}h(s') - h(s)\} \leq c(s,d), \forall d \in \mathcal{D}_s, \forall s \in \mathcal{S} \tag{4.7b}$$

$$g \text{ and } h(s) \; unconstrained \tag{4.7c}$$

Note that, if $(g^*,\ h^*)$ is an optimal solution of (4.7), then g^* is the long-run average cost under an optimal policy and $h^* + ke$ is the bias vector under the same optimal policy for some constant k. Together, this gain and bias form a solution to the optimality equation. However, we find it more intuitive to analyze this model trough its dual formulation since the dual decision variables directly relate to states and decision rules in the MDP. $x(s,d)$ is the dual decision variable, taking positive value if action d is taking in state s in steady state.

$$\min \quad \sum_{s \in \mathcal{S}} \sum_{d \in \mathcal{D}_s} c(s,d)x(s,d) \tag{4.8a}$$

$$\text{s.t.} \quad \sum_{d \in \mathcal{D}_{s'}} x(s',d) - \sum_{s \in \mathcal{S}} \sum_{d \in \mathcal{D}_s} P\{s'|s,d\}x(s,d) = 0, \forall s' \in \mathcal{S} \tag{4.8b}$$

$$\sum_{s \in \mathcal{S}} \sum_{d \in \mathcal{D}_s} x(s,d) = 1 \tag{4.8c}$$

$$x(s,d) \geq 0, \forall d \in \mathcal{D}_s, \forall s \in \mathcal{S} \tag{4.8d}$$

Observe that the primal LP (4.7) has $|\mathcal{S}|$ columns and $\sum_{s \in \mathcal{S}} |\mathcal{D}_s|$ rows, while the dual LP (4.8) has $|\mathcal{S}|$ rows and $\sum_{s \in \mathcal{S}} |\mathcal{D}_s|$ columns. In addition to the above discussion, this is another motivation to work on the dual formulation.

In the dual LP (DLP) (4.8) model, the decision variable $x(s,d)$ can be interpreted as the stationary probability that the system in steady state is in state s; $x(s,d) = \sum_{t=0}^{\infty} P(s^t, d)$, such that (4.8b) and (4.8c) provide the system of equations (e.g. $\Pi P = \Pi$, $\sum_{s \in \mathcal{S}} \pi(s) = 1$) to find the corresponding stationary distribution.

Each basic feasible solution to DLP corresponds to a policy in the MDP. Let x be a basic feasible solution, then $\pi_x(s) = d$ if $x(s,d) > 0$. Note that unichain models have possibly an empty set of transient states, then $\pi_x(s)$ can be chosen arbitrarily for transient state s with $x(s,d) = 0$ for all $d \in \mathcal{D}_s$.

Our MDP model grows exponentially in the length of customer order horizon and the maximum number of arrivals can be observed over a period. As a result, using the DLP model directly creates significant computational limitations to solve the problem, even it provides cost optimal customer admission and capacity allocation policies when the system is in steady-state. The reminder of this chapter consists of our methodology, to overcome this computational difficulty and solve larger instances in shorter amount of time.

4.2 Action Space Reduction: Structural Results

In this section, we partially characterize the structure of the optimal policies. We first analyze the properties of the MDP model and the optimal solution behaviors, while we attempt to find out the necessary conditions that a set of valid actions should satisfy for any possible state $s^t \in \mathcal{S}$. Our very first effort is in characterizing the suboptimal actions when the length of the customer order horizon is 2, i.e., $K = 2$. Then, we generalize these results for longer customer order horizons.

To avoid trivial cases in the solutions, we make the following assumptions on the cost parameters.

Assumption 1. $c_2^e < c_1^e$*: serving a high-priority job request early is costlier than serving a low-priority job request early.*

Assumption 2. $c_r < c_o$*: rejection of a low-priority job request incurs less cost than its fulfillment in overtime.*

Assumption 2 ensures that the rejection decision remains relevant in the problem.

In the following part, we present our structural results based on these assumptions. Our initial results are derived for systems with $K = 2$.

Remark. *For system with $K = 2$, under Assumption 1, if some customers have to be served early, the server provider starts with the low-priority customers first.*

Proposition 1. *When $c_2^e < c_1^e < c_r < c_o$, any action $d = (r,\ y)$ in a state $s^t \in \mathcal{S}$ that does not satisfy the following conditions is suboptimal.*

$$min\{[(x_{10} + a_{10} + a_{20}) - M]^+, a_{20}\} \leq r_{20} \leq a_{20} \tag{i}$$

$$min\{[[(a_{11} + a_{21}) - M]^+ - y_{21}]^+, a_{21}\} \leq r_{21} \leq [a_{21} - [M - (x_{10} + a_{10} + a_{20} - r_{20})]^+]^+ \tag{ii}$$

$$y_{10} = x_{10} + a_{10} \tag{iii}$$

$$\begin{aligned} &y_{11} = 0 \text{ if } y_{21} < min\{[M - (x_{10} + a_{10} + a_{20} - r_{20})]^+, (a_{21} - r_{21})\} \\ &y_{11} \leq min\{[M - (x_{10} + a_{10} + a_{20} - r_{20} + y_{21})]^+, a_{11}\},\ otherwise \end{aligned} \tag{iv}$$

$$y_{20} = a_{20} - r_{20} \tag{v}$$

$$y_{21} \leq min\{[M - (x_{10} + a_{10} + a_{20} - r_{20})]^+, (a_{21} - r_{21})\} \tag{vi}$$

with $y_{11} + y_{21} \leq [M - (x_{10} + a_{10} + a_{20} - r_{20})]^+$.

Proof. (iii) and (v) are ensured by the definition of actions, and notice that these conditions appear in all following propositions.

(i): After observing arrivals, if no rejection is made, $[(x_{10} + a_{10} + a_{20}) - M]^+$ is the number of requests that need to be fulfilled in overtime operation. In that case, the corresponding overtime cost is $c_o[(x_{10} + a_{10} + a_{20}) - M]^+$. However, if one of the excess low-priority job requests is rejected , the corresponding cost is $c_r + c_o([(x_{10} + a_{10} + a_{20}) - M]^+ - 1)$ which is less than the overtime cost. It is easy to see that the total rejection and overtime cost is decreasing in the number of rejections since $c_r < c_o$. Therefore, whenever it is certain that the system operates in overtime, the controller

should reject the excess arriving low-priority job requests. So, the minimum number of necessary rejections determines the lower bound for r_{20}. The upper bound on r_{20} follows by the definition.

(iii)-(v): All high-priority job requests for t should be fulfilled in the current time period. Similarly, all admitted all low-priority job requests for t should be fulfilled in the current time period.

(ii) and (vi) together mean that if there exist some excess capacity, the controller first decides to early serve excess low-priority job requests for time $t+1$, then decides whom to reject among remaining jobs. Since $c_r < c_o$, we should reject arriving low-priority job requests whenever we are sure that they will cause overtime. $a_{11} + a_{21}$ is the number of arriving request for $t+1$. If we accept all, in the next period they will all become high-priority customers who want to be served immediately. If $a_{11}+a_{21} > M$, then we know that there will be a need for overtime in the next period. Therefore, to avoid this upcoming cost, we should reject the excess low-priority job requests and incur c_r in this period instead of c_o. Now, considering $c_2^e < c_r$, rather than rejecting the excess orders, we should serve them early if there is some available capacity. Therefore, if there is some available capacity, we should reject at least $min\{[[(a_{11} + a_{21}) - M]^+ - y_{21}]^+, a_{21}\}$ among low-priority job requests due $t+1$. The upper bound can be found directly, since arriving requests can be fulfilled up the amount of remaining available capacity. Similarly, the upper bound for y_{21} is followed by definition.

(iv) forces the controller to early serve among low type orders first, then start to serve high type ones if he serves any customers early. We use the same arguments as in the discussion of previous conditions; $a_{11} + a_{21} - r_{21}$ is the number of arriving high-priority job requests due $t+1$ and in the next period they will all want to be served immediately. If at this decision period there exist some available capacity and $a_{11} + a_{21} - r_{21} > M$, then we are sure that we must serve some orders early to avoid the overtime cost in the next period. Since $c_2^e < c_1^e$, we should first serve low type customers to incur c_2^e instead of c_1^e. The upper bound on y_{11} is intuitive, settled by definition. □

Proposition 2. *When $c_r < c_2^e < c_1^e < c_o$, any action $d = (r,\ y)$ in a state $s^t \in \mathcal{S}$ that does not satisfy the following conditions is suboptimal.*

$$min\{[(x_{10} + a_{10} + a_{20}) - M]^+, a_{20}\} \leq r_{20} \leq a_{20} \quad \text{(i)}$$

$$min\{[(a_{21} + a_{21}) - M]^+, a_{21}\} \leq r_{21} \leq a_{21} \quad \text{(ii)}$$

$$y_{10} = x_{10} + a_{10} \quad \text{(iii)}$$

$$y_{11} \leq min\{[M - (x_{10} + a_{10} + a_{20} - r_{20} + y_{21})]^+, a_{11}\} \quad \text{(iv)}$$

$$y_{20} = a_{20} - r_{20} \quad \text{(v)}$$

$$\begin{aligned} &y_{21} = 0,\ \ if\ r_{21} > 0 \\ &y_{21} \leq min\{[M - (x_{10} + a_{10} + a_{20} - r_{20})]^+, a_{21}\},\ otherwise \end{aligned} \quad \text{(vi)}$$

with $y_{11} + y_{21} \leq [M - (x_{10} + a_{10} + a_{20} - r_{20})]^+$.

Proof. (iii),(iv), and (v) are ensured by definition. (i) is proved in Proposition 1 , and the same proof is still valid since $c_r < c_o$.

(ii): Since $c_r < c_e < c_o$, the system should reject, rather than serve early, arriving low-priority requests whenever it is certain that they will cause overtime. $a_{11} + a_{21}$ is the number of arriving request due $t + 1$. If all job requests for $t + 1$ are admitted and held in the queue, in the next period they will all become high-priority customers have to be served immediately. If $a_{11} + a_{21} > M$, then it is known that there will be a need for overtime for sure. Therefore, to avoid this upcoming cost, the excess low-priority job requests should be rejected and the system incurs $c_r r_{21}$ in this period instead of $c_o r_{21}$ in the next period. At least $min\{[(a_{11} + a_{21}) - M]^+, a_{21}\}$ among low-priority job requests for $t + 1$ should be rejected. The upper bound on r_{21} is followed by definition.

$(iii) - (v)$: All high-priority job requests for t should be fulfilled in the current time period. Similarly, (v) all admitted all low-priority job requests for t should be fulfilled in the current time period.

(iv): The upper bound on the early service decision of high-priority job requests is easy to show. Due to Assumption 2 together with conditions (iii) and (iv), the idle

capacity is $[M-(x_{10}+a_{10}+a_{20}-r_{20}+y_{21})]^+$. Due to the dynamic priority rule, we have $x_{11}=0$ and the system can only serve newly arriving requests, up to the amount of the idle capacity.

(vi) and (vii) should be assessed together, and in relation to (ii). (ii) and (vi) together imply that if at least one low-priority request for $t+1$ is rejected, none of admitted low-priority requests is served early. Certainly, the upper bound for the rejection and also for the early service decision is the number of arriving low-priority requests, a_{21}. Due to (ii), at least as many orders as the number of low-priority orders that cause overtime in the next period are rejected. Since $c_r < c_2^e$, rather than serving a low-priority request early, the system should reject it, and incur less cost. □

Proposition 3. *When $c_r < c_o < c_2^e < c_1^e$, any action $d=(r,\ y)$ in a state $s^t \in \mathcal{S}$ that does not satisfy the following conditions is suboptimal.*

$$min\{[(x_{10}+a_{10}+a_{20})-M]^+, a_{20}\} \le r_{20} \le a_{20} \quad \text{(i)}$$

$$min\{[(a_{21}+a_{21})-M]^+, a_{21}\} \le r_{21} \le a_{21} \quad \text{(ii)}$$

$$y_{10} = x_{10}+a_{10} \quad \text{(iii)}$$

$$y_{11} = 0 \quad \text{(iv)}$$

$$y_{20} = a_{20}-r_{20} \quad \text{(v)}$$

$$y_{21} = 0 \quad \text{(vi)}$$

Proof. (iii) and (v) are ensured by definition. (i) is proved in Proposition 1 and (ii) is proved in Proposition 2 and these proofs still hold since $c_r < c_o < c_2^e < c_1^e$.

$(iii)-(v)$: All high-priority job requests for time t should be fulfilled in the current time period. Similarly, all admitted all low-priority job requests for t should be fulfilled in the current time period.

(iv): Since $c_o < c_2^e < c_1^e$, all arriving requests for $t+1$ should wait in the queue until the next period. $a_{11}+(a_{21}-r_{21})$ is the total number of arriving requests for $t+1$, and it is certain that $a_{11}+(a_{21}-r_{21}) \le M$, due to (ii). Therefore, it is obvious that these job requests do not cause overtime in the next period. Independent of the upcoming arrivals, in the presence of idle capacity, if one of these high-priority job requests is

served in advance, the system incurs c_1^e amount of early service cost. After observing arrivals, assume that $a_{11}+(a_{21}-r_{21})-1+a_{10}^{t+1}+(a_{20}^{t+1}-r_{20}^{t+1})>M$, and the system incurs $c_o[(a_{11}+(a_{21}-r_{21})-1+a_{10}^{t+1}+(a_{20}^{t+1}-r_{20}^{t+1}))-M]^+$. If the system does not serve any request early, at time $t+1$ we incur $c_o[(a_{11}+(a_{21}-r_{21})+a_{10}^{t+1}+(a_{20}^{t+1}-r_{20}^{t+1}))-M]^+$, which is clearly less than $c_1^e+c_o[(a_{11}+(a_{21}-r_{21})-1+a_{10}^{t+1}+(a_{20}^{t+1}-r_{20}^{t+1}))-M]^+$. The observation still holds when $a_{11}+(a_{21}-r_{21})-1+a_{10}^{t+1}+(a_{20}^{t+1}-r_{20}^{t+1})<M$ after the arrivals. Therefore, even if in the presence of idle capacity, the controller should not serve early. (vi) follows by the same argument. □

These results identify a set of suboptimal policies based only on system parameters such as server capacity and costs without relying on the arrival probabilities. However, these propositions are based on systems with $K=2$. Addressing more general systems, we extend some of these results to larger customer order horizons.

Definition 1. *A state $s^t \in \mathcal{S}$ at time t is a j-boundary state, if there are at least M job requests due $t+j$, $j=0,\dots,K-1$ in the system, including the job requests that are already waiting in the queue and new arrivals prior to any rejection.*

Proposition 4. *Any action $d=(r,\ y)$ in a j-boundary state $s^t \in \mathcal{S}$, $j=0,\dots,K-1$ that does not satisfy the following conditions is suboptimal.*

$$min\{[(x_{10}+a_{10}+a_{20})-M]^+, a_{20}\} \le r_{20} \le a_{20} \quad \text{(i)}$$

$$min\{[(x_{1j}+x_{2j}+a_{1j}+a_{2j})-M]^+, a_{2j}\} \le r_{2j} \le a_{2j}\ \forall j, j=1,\dots,K-1 \quad \text{(ii)}$$

$$y_{10}=x_{10}+a_{10} \quad \text{(iii)}$$

$$y_{1j}=0\ \forall j, j=1,\dots,K-1 \quad \text{(iv)}$$

$$y_{20}=a_{20}-r_{20} \quad \text{(v)}$$

$$y_{2j}=0\ \forall j, j=1,\dots,K-1 \quad \text{(vi)}$$

Proof. (iii), and (v) are ensured by definition. (i) is proved in Proposition 1, and the same argument is still valid due to Assumption 2.

The lower bound in (ii) can be proved using the same arguments as in the case with $K=2$. Since the state is j-boundary, $j=0,\dots,K-1$, the server can not serve

early any requests over the planning horizon. According to Assumption 2, rejection of excess job requests incurs less when there is overtime for sure. The upper bound on the rejection decision (ii) follows by definition since the controller can only reject among the arriving requests.

$(iii) - (v)$: All high-priority job requests for t should be fulfilled in the current time period. Similarly, all admitted all low-priority job requests for t should be fulfilled in the current time period.

Early service decisions in (iv) and (vi) are easy to show. The system has already worked on either in full capacity or in overtime since the state is 0-boundary; hence there does not exist any idle capacity that can be used for early service. □

Proposition 4 is the least restrictive and the most intuitive result on suboptimal actions, when the server capacity is relatively scarce for K and A.

Proposition 5. *When $c_2^e < c_1^e < c_r < c_o$, any action $d = (r,\ y)$ in a state $s^t \in \mathcal{S}$, that does not satisfy the following conditions is suboptimal.*

$$min\{[(x_{10} + a_{10} + a_{20}) - M]^+, a_{20}\} \leq r_{20} \leq a_{20} \tag{i}$$

$$\begin{aligned} &min\{[overtime_j - y_{2j}]^+, a_{2j}\} \leq r_{2j} \leq a_{2j}\ for\ j = 1, \ldots, K-1\ if\ jc_2^e \leq c_r \\ &min\{overtime_j, a_{2j}\} \leq r_{2j} \leq a_{2j}\ for\ j = 1, \ldots, K-1\ otherwise \end{aligned} \tag{ii}$$

$$y_{10} = x_{10} + a_{10} \tag{iii}$$

$$y_{1j} \leq min\{remain_1, x_{1j} + a_{1j}\},\ for\ j = 1, \ldots, K-1 \tag{iv}$$

$$y_{20} = a_{20} - r_{20} \tag{v}$$

$$y_{2j} \leq min\{remain_2, (x_{2j} + a_{2j} - r_{2j})\}\ for\ j = 1, \ldots, K-1 \tag{vi}$$

where $overtime_j = [[(x_{1j} + x_{2j} + a_{1j} + a_{2j}) - M]^+$, $remain_1 = [M - (y_{10} + y_{20} + \sum_{j=1}^{K-1} y_{2j} + \sum_{k=0}^{j-1} y_{1k})]^+$ $remain_2 = [M - (y_{10} + y_{20} + \sum_{k=0}^{j-1} y_{2k})\ )]^+$ *with* $\sum_{j=1}^{K-1} y_{1j} + y_{2j} \leq [M - (x_{10} + a_{10} + a_{20} - r_{20})]^+$

Proof. (iii) and (v) are ensured by the definition of actions. Condition (i) is shown in Proposition 1 and due to Assumption 2, the argument is still relevant.

(ii): This condition ensures early service of low-priority job requests that would definitely cause overtime at their preferred service completion time $t + j$, rather than their rejection when $jc_2^e < cr$. Note that $a_{1j} + a_{2j}$ is the number of arriving request due $t + 1$, and the total number of job request for $t + j$ is $x_{1j} + x_{2j} + a_{1j} + a_{2j}$. If all job requests for $t + j$ are admitted and held in the queue, in j periods they will all become high-priority requests that have to be served in current period. If $x_{1j} + x_{2j} + a_{1j} + a_{2j} > M$, then it is known that there will be a need for overtime for sure at $t + j$. Therefore, to avoid this overtime cost, the excess low-priority job requests should be rejected and the system incurs $c_r r_{2j}$ in this period instead of $c_o r_{2j}$ in j periods later. At least $min\{[(x_{1j} + x_{2j} + a_{1j} + a_{2j}) - M]^+, a_{2j}\}$ among low-priority job requests for $t + j$ should be rejected. The upper bound on r_{2j} follows by definition.

$(iii) - (v)$: All high-priority job requests for t should be fulfilled in the current time period. Similarly, all admitted all low-priority job requests for t should be fulfilled in the current time period.

(iv): This condition is easy to show using the definition of actions. Due to Assumption 1 and together with conditions (iii), (v), and (vi), the idle capacity is $[M - (x_{10} + a_{10} + a_{20} - r_{20} + \sum_{j=1}^{K-1} y_{2j})]^+$. It is known that early service can only occurs in regular capacity, thus job requests in the system can be served at most the amount of remaining idle capacity. Similarly, the upper bound on y_{2j} is settled by definition. □

Proposition 6. *When $c_r < c_2^e < c_1^e < c_o$, any action $d = (r,\ y)$ in a state $s^t \in \mathcal{S}$ that does not satisfy the following conditions is suboptimal.*

$$min\{[(x_{10} + a_{10} + a_{20}) - M]^+, a_{20}\} \leq r_{20} \leq a_{20} \tag{i}$$

$$min\{overtime_j, a_{2j}\} \leq r_{2j} \leq a_{2j}\ for\ j = 1, \ldots, K-1 \tag{ii}$$

$$y_{10} = x_{10} + a_{10} \tag{iii}$$

$$y_{1j} \leq min\{remain_1, x_{1j} + a_{1j}\},\ for\ j = 1, \ldots, K-1 \tag{iv}$$

$$y_{20} = a_{20} - r_{20} \tag{v}$$

$$y_{2j} \leq min\{remain_2, (x_{2j} + a_{2j} - r_{2j})\}\ for\ j = 1, \ldots, K-1 \tag{vi}$$

where $overtime_j = [[(x_{1j} + x_{2j} + a_{1j} + a_{2j}) - M]^+$, $remain_1 = [M - (y_{10} + y_{20} + \sum_{j=1}^{K-1} y_{2j} + \sum_{k=0}^{j-1} y_{1k})]^+$, *and* $remain_2 = [M - (y_{10} + y_{20} + \sum_{k=0}^{j-1} y_{2k}))]^+$ *with* $\sum_{j=1}^{K-1} y_{1j} + y_{2j} \leq [M - (x_{10} + a_{10} + a_{20} - r_{20})]^+$

Proof. (iii),(iv), and (v) are ensured by definition. (i) is proved in Proposition 1 , and the same proof is still valid since $c_r < c_o$.

(ii): Since $c_r < c_2^e < c_o$, the system should first reject excess arriving low-priority job requests whenever it is certain that they will cause overtime at $t+j$, $j = 1, \ldots, K-1$. Note that $a_{1j} + a_{2j}$ is the number of arriving request due $t+1$, and the total number of job request for $t + j$ is $x_{1j} + x_{2j} + a_{1j} + a_{2j}$. If all job requests for $t + j$ are admitted and held in the queue, in j periods they will all become high-priority customers have to be served in current period. If $x_{1j} + x_{2j} + a_{1j} + a_{2j} > M$, then it is known that there will be a need for overtime for sure at $t + j$. Therefore, to avoid this overtime cost, the excess low-priority job requests should be rejected and the system incurs $c_r r_{2j}$ in this period instead of $c_o r_{2j}$ in $t+j$. At least $min\{[(x_{1j} + x_{2j} + a_{1j} + a_{2j}) - M]^+, a_{2j}\}$ among low-priority job requests for $t + j$ should be rejected. The upper bound on r_{2j} follows by definition.

$(iii) - (v)$: All high-priority job requests for t should be fulfilled in the current time period. Similarly, (v) all admitted all low-priority job requests for t should be fulfilled in the current time period.

(iv): The upper bound on the early service decision of high-priority job requests is easy to show. Due to assumption 2 together with conditions (iii) and (iv), the idle capacity is $[M - (y_{10} + y_{20} + \sum_{j=1}^{K-1} y_{2j} + \sum_{k=0}^{j-1} y_{1k})]^+$. It is known that early service can only occurs in regular capacity, thus job requests in the system can be served at most the amount of remaining idle capacity. Similarly, the upper bound on y_{2j} is settled by definition. □

Proposition 7. *When $c_r < c_o < c_2^e < c_1^e$, any action $d = (r,\ y)$ in a state $s^t \in \mathcal{S}$ that does not satisfy the following conditions is suboptimal.*

$$min\{[(x_{10} + a_{10} + a_{20}) - M]^+, a_{20}\} \leq r_{20} \leq a_{20} \quad \text{(i)}$$

$$min\{[(x_{1j} + x_{2j} + a_{1j} + a_{2j}) - M]^+, a_{2j}\} \leq r_{2j} \leq a_{2j}\ for\ j = 1, \ldots, K-1 \quad \text{(ii)}$$

$$y_{10} = x_{10} + a_{10} \quad \text{(iii)}$$

$$y_{1j} = 0\ for\ j = 1, \ldots, K-1 \quad \text{(iv)}$$

$$y_{20} = a_{20} - r_{20} \quad \text{(v)}$$

$$y_{2j} = 0\ for\ j = 1, \ldots, K-1 \quad \text{(vi)}$$

Proof. (iii) and (v) are ensured by definition. (i) is proved in Proposition 1 and (ii) is proved in Proposition 6, the proofs are valid, since $c_r < c_o < c_2^e < c_1^e$.

$(iii) - (v)$: All high-priority job requests for t should be fulfilled in the current time period. Similarly, all admitted all low-priority job requests for t should be fulfilled in the current time period.

(iv): Since $c_o < c_2^e < c_1^e$, all arriving requests for $t + j$, $j = 1, \ldots, K-1$ should be waited in the queue until their preferred service completion time. For instance, $a_{1j} + (a_{2j} - r_{2j})$ is the total number of arriving requests for $t + j$, and it is certain that $x_{1j} + x_{2j} + a_{1j} + (a_{2j} - r_{2j}) \leq M$, according to (ii). Therefore, at time $t + j$ these job requests do not cause overtime. Independent of the upcoming arrivals, in the presence of idle capacity, if one of these high-priority job requests is served in advance, the system incurs jc_1^e amount of early service cost. For $j = 1$ refer to Proposition 3. For $j > 1$ since $c_o < c_1^e$, the same argument still holds. Early service of any job requests incurs more than overtime. Therefore, even if in the presence of idle capacity, the controller should not serve early. (vi) is followed by the same argument. □

In short, we first analyze the structural properties of the optimal solution and reveal some necessary conditions for optimality. Integrating these results into our mathematical model decreases the size of the action space significantly, where the DLP search for optimal policy, and strengthen the performance of the column-based solution method. However, as the customer order horizon K increases, the simultaneous rejection and early service decisions become significantly interdependent, which may lessen the impact of our propositions. Therefore, in addition to these partial characterization results, we also develop state space reduction methods.

4.3 State Space Reduction

The DLP model provides cost-optimal customer admission and capacity allocation policies when the infinite time horizon is assumed to be in steady-state. Despite the useful and practical insights it provides for the system controller, it is only tractable when the rolling planning horizon is very short, and customer arrivals and the server capacity are very limited. Thus, in this section we introduce new techniques to overcome the computational limitations inherited from the exponentially growing state space. These new methods are based on generating a partition of an MDP to construct a minimal MDP which can be used to solve the original MDP. We use exact and approximate methods, introduced respectively.

4.3.1 Decomposition of State Space

Long-run average cost criterion is generally associated with the steady-state behavior of MDP. Therefore, the classification of the states (e.g., recurrent, transient) becomes important. As mentioned in Section 4.1, solving the DLP provides alternative optimal policies. These policies differ in the actions that are taken in transient states since they have no impact into cost when the system is in steady-state. In short, the DLP reveals the optimal decision only for recurrent states. With this in mind, we attempt to narrow down the state space by removing the transient states under all deterministic policies. For this purpose, we employ an exact decomposition algorithm for unichain classification problem which was first established by Bather [29], then improved by

Ross and Varadarajan [30]. The algorithm is developed for finite state space and finite action space MDPs, based on the accessibility between the states and aims to find out the states that are recurrent under all deterministic stationary policies.

The method for the decomposition creates a directed graph G^2 using the state space $\mathcal{S}$ as the vertex set. G^2 has an arc (s, s') between states s and s' if at least one of the Markov chains of the MDP has a positive one-step transition from s to s', i.e., $P(s'|s, d) > 0$ for at least one action $d \in \mathcal{D}_s$. Through the decomposition, at each step, the state space is divided into several levels composed of (i) the closed, communicating subsets of the state space, hence strongly connected components of G^2, (ii) the open strongly connected components of G^2. The algorithm distinguishes the states from which for any deterministic policy absorption in a strongly connected component occurs certainly, the set of transient states T, and the states from which with an appropriate choice of the policy the absorption can be avoided, the set of recurrent states R. This decomposition algorithm is formalized in detail in [11]. [11] also shows that the algorithm finds the set of transients states under all deterministic policies denoted by T, in polynomial time and its complexity is $O\left(A \cdot N^2\right)$.

Once the set of transient states under all deterministic policies (T) is determined, it can be removed from the state space $\mathcal{S}$. At each iteration of the algorithm, while detecting the transient and recurrent states, it also eliminates some actions that can not avoid the absorption in the recurrent class, i.e. $s \in R$ and $\mathcal{D}_s = \mathcal{D}_s \setminus d : \sum_{s' \in T} P(s'|s, d) > 0$.

A reduced MDP (RMDP) is defined on the reduced state space $\mathcal{S}'$, such that $\mathcal{S}' = \mathcal{S} \setminus T$, and the reduced action space $\mathcal{D}' \subseteq \mathcal{D}$, which is restricted respect to above elimination rule. Any deterministic policy in the original MDP also defines a deterministic policy in RMDP. Since the states in T are always transient in the original MDP, the recurrent classes for these two Markov processes coincide. Therefore, the long-run average cost, defined in terms of stationary probabilities of recurrent states and the cost function, remains the same.

To solve the problem optimally, we also use the DLP model of the RMDP. Notice that the DLP model of RMDP has $|\mathcal{S}' = \mathcal{S} \setminus T|$ rows and $|\mathcal{D}'|$ columns. Thus, when $T \neq \emptyset$, we reduce the problem of finding the optimal joint admission and capacity allocation policy to a smaller problem.

4.3.2 State Aggregation and Aggregate MDP Model

The use of aggregation techniques is another common approximate approach to reduce the complexity related to the large state space. The key principle of this approach is to cluster the original large state space into aggregate subgroups, which are treated as newly created states in the aggregate MDP. Therefore, as the number of states reduces, the size of the probability transition matrix decreases. Using this approach, the original MDP can be represented by a smaller approximate MDP. Since the approximate model preserves the Markovian property, any solution method for MDPs, such as LP, is still valid. Besides, the optimal policy for the aggregate problem is an approximate solution for the original problem, and via evaluation algorithms we can easily evaluate the approximate optimal policy on the original MDP's space.

In this subsection, we introduce two different state aggregation methods, which use action and reward structure to judge whether a variable should be included in an aggregate state or not. Briefly, the first method clusters ε-equivalent states, that we define later in this chapter.The second method is a problem-specific approach that only takes into account structural properties of our system. Each method provides different partitions of $\mathcal{S}$ and results in different approximate optimal policies. In the following two chapters we introduce each methods in details, then we discuss their advantages and disadvantages.

4.3.2.1 ε-Aggregation Method

One general way of generating the reduced MDP is to partition the state space $\mathcal{S}$ by clustering "equivalent" states into meta-states. In this subsection, we first state the necessary conditions to define two distinct states $s, s' \in \mathcal{S}$ to be equivalent. Then, we generalize equivalency conditions and define ε-equivalent states.

The equivalence between states, is measured in terms of transition probabilities and cost streams. Equivalent states are assumed to behave approximately the same under all policies.

Definition 2. *Given an MDP, described with tuple* $< \mathcal{S}, \mathcal{D}, P, c >$*, where* $\mathcal{S}$*,* $\mathcal{D}$*,* P *and* c *stand for state space, action space, transition probability distribution and immediate cost function, respectively, two states* $s, s' \in \mathcal{S}$ *are equivalent if for all* $d \in \mathcal{D}$ *following conditions are satisfied.*

$$c(s,d) = c(s',d) \quad \textit{and} \quad P(s''|s,d) = P(s''|s',d),\ \forall s'' \in \mathcal{S} \tag{4.7}$$

The Definition 2 assumes that a general action space $\mathcal{D}$ exists and every action $d \in \mathcal{D}$ is allowable in all states. However, in our MDP model not every action can be taken in all states. The set of allowable actions in a state is denoted by $\mathcal{D}_s$ and the action space is the union of $\mathcal{D}_s$s, i.e. $\cup_{s\in\mathcal{S}}\mathcal{D}_s$. Besides, this definition should be generalized for MDPs in which each state has a distinct and individual set of actions.

Definition 3. *Given an MDP, described with tuple* $< \mathcal{S}, \mathcal{D}, P, c >$*, two states* $s, s' \in \mathcal{S}$ *with the set of actions* $\mathcal{D}_s, \mathcal{D}_{s'} \subset \mathcal{D}$*, respectively, are equivalent if for all* $d \in \mathcal{D}_s$ *there is a* $d' \in \mathcal{D}_{s'}$ *such that*

$$c(s,d) = c(s',d') \quad \textit{and} \quad P(s''|s,d) = P(s''|s',d'),\ \forall s'' \in \mathcal{S} \tag{4.8}$$

The simple action-sequence equivalence given in Definition 3 is introduced by [31], and a useful tool used in state aggregation procedures. For most of the MDPs, finding equivalent states using 3 is also computationally expensive, since it provides the smallest possible state space. Therefore, researchers introduce different equivalence and similarity relations. One of these approaches is ε-homogeneity technique, the milestone of ε-aggregation method. This technique is first introduced by Dean et al. [32] to the artificial intelligence community, and an analog of it has been released by Ortner [33]. [32] specifically studies the resulting approximate optimal policies of the proposed method for bounded parameter MDPs, the generalization of exact MDPs, which are defined by giving upper and lower bounds on transition probabilities and rewards. [33] tries to convert the similarity idea, that lay behind, by setting a pseudometric on the state space. The use of ε-homogeneity leads the partition of state

space $\mathcal{S}$, let say ε-partition, of an MDP into disjoint set of states. From definition of ε-homogeneity given in 4, we derive definition of ε-equivalent states.

Definition 4. *For fixed $\varepsilon > 0$, and positive coefficients $c_c, c_p > 0$, an ε-partition of the state space $\mathcal{S}$ is a minimal partition of $\mathcal{S}$ into aggregated states $S_1, \ldots, S_k$ such that for s, $s' \in S_i$, and $S_j \cap S_i = \emptyset$*

$$c_c \left\| c(s,d) - c(s',d') \right\| < \varepsilon \quad \textit{and} \quad c_p \left\| \sum_{s'' \in S_j} p(s''|s,d) - \sum_{s'' \in S_j} p(s''|s',d') \right\| < \varepsilon, \quad \forall j \neq i \tag{4.9}$$

Definition 5. *Given an MDP, described with tuple $< \mathcal{S}, \mathcal{D}, P, c >$, a fixed $\varepsilon > 0$ and positive coefficients $c_c, c_p > 0$, two states $s, s' \in \mathcal{S}$ with their set of actions $\mathcal{D}_s, \mathcal{D}_{s'} \subset \mathcal{D}$, respectively, are ε-equivalent if for all $d \in \mathcal{D}_s$ there is a $d' \in \mathcal{D}_{s'}$ such that*

$$c_c \left\| c(s,d) - c(s',d') \right\| < \varepsilon \quad \textit{and} \quad c_p \left\| \sum_{s'' \in \mathcal{S} \setminus \{s,s'\}} P(s''|s,d) - \sum_{s'' \in \mathcal{S} \setminus \{s,s'\}} P(s''|s',d') \right\| < \varepsilon \tag{4.10}$$

ε-aggregation method generates ε-partition of $\mathcal{S}$. The pseudo code of the method is given in appendix A. Once all states $s \in \mathcal{S}$ are generated, with their set of actions $\mathcal{D}_s$, they are sorted in increasing order according to the number of actions. The first state is assigned as the seed state of the first cluster, so the first meta-state. An unassigned state that is a candidate state is assigned to an existing cluster if it is ε-equivalent of the corresponding seed. If it is not assigned to any meta-state, it is assigned as the new meta-state's seed. This process continues until there are no unassigned states. Considering the ease of implementation, we first generate meta-states according to cost condition in 4.10. Then, if there are states that do not ensure the probability condition in 4.10 in these meta-states, the status of these states are changed to unassigned, and are reassigned to existing appropriate meta-state according to the Definition 5.

Once an ε-partition $\{S_1, \ldots, S_n\}$ where $\bigcup_{i=1}^{n} S_i = \mathcal{S}$ and $S_k \cap S_m = \emptyset$ if $k \neq m$, is found, for each $i \in 1, \ldots, N$, the set of actions available in state S_i in the aggregate MDP, $\mathcal{D}_{S_i}$ is defined as $\cap_{s \in S_i} \mathcal{D}_s$, to avoid from state-action mismatches.

It is certain that the accuracy of the aggregate MDP depends on ε, which is employed in construction of partition. In theory, when $\varepsilon = 0$ we have the intractable full original MDP, and the original optimal policy. As ε increases, the number of states in the aggregate MDP decreases significantly, and we have a smaller aggregate MDP model.

The ε-aggregation method itself takes more time and requires more computations since it calculates immediate costs and transition probabilities for each state-action pair to realize related comparisons. However, with our three step approach, expressed in Appendix A, we also enhance its computations.

4.3.2.2 Total Job Aggregation Method

Total job aggregation method is a problem specific approach. Its pseudo code is also provided in Appendix A. In the most clear way, this aggregation method looks at pre-decision states, just after realization of arrivals, just before any rejection or service decisions, stats with equal sum of current queue and arrival elements are the same meta-state. That is two states $s = (x, a), s' = (x', a') \in \mathcal{S}$ are assigned to the same meta-state if $x + a = x' + a'$.

This aggregation method makes use of structural properties given in problem definition. For states, where the total number of jobs is equal there exists some actions, which the one-step transition probabilities match. When the system capacity is scarce, the immediate cost of taking these actions in these states would also be equal, since overtime and rejection are inevitable. When the capacity is large, we enlarge the definition of equivalence and we only consider the transition probability matches. For moderate capacity, there always exists some actions which ensures the conditions given in Definition 3.

Example: Consider a state s when $K = 3$ and $A = 2$.

$$s = (x, a) = [(1, 2, 0, 0, 2, 0), (1, 0, 0, 2, 1, 1)]$$

Let the service capacity be $M = 3$. Then, s is a 0-boundary state, since $x_{10} + a_{10} + a_{20} > 3$. Therefore, an optimal action in s, can not serve any request for the next periods in advance. Now consider a second state $s' = (x', a') = [(2, 2, 0, 0, 1, 0), (0, 0, 0, 2, 2, 1)]$ under the same problem settings. As s, s' can also be called 0-boundary and has the same amount of total jobs in the queue. Respect to our second aggregation technique we can assign these two states into the same meta-state, since their number of total jobs are equal and for action $d = (r, y) = [(1, 0, 0), (3, 0, 0, 0, 0, 0)]$ which is allowable in s and s' the one-step transition probabilities and immediate costs match. Now, let the service capacity be $M = 5$. That is, some jobs can be served early, while rejection and overtime are still in consideration. Consider another state $s'' = (x'', a'') = [(0, 0, 0, 0, 1, 0), (2, 2, 0, 2, 2, 1)]$. These two states can also be assigned to the same meta-states, due to same argument: their number of total jobs are equal and for action $d = (r, y) = [(0, 0, 0), (2, 0, 0, 2, 0, 0)]$ which is allowable in s and s'' the one-step transition probabilities and immediate costs match.

This aggregation method provides a great advantage in terms of computation time. It does not take into account the action dependent cost and transition probabilities, thus it requires less computations while generating new states for the aggregate MDP model. Although, it is advantageous as a fast-acting offline aggregation method for intractable problems, the state space reduction trough it is limited on the values of K and M. For same K and A, it generates the same meta-states.

4.3.2.3 Aggregate MDP Model

Each aggregation method partitions the state space into clusters of states $\{S_1, \ldots, S_N\}$. Remember that, for each $i \in 1, \ldots, N$, the set of actions available in state S_i in the

aggregate MDP, $\mathcal{D}_{S_i}$ is defined as $\cap_{s \in S_i} \mathcal{D}_s$. Using the fixed-weight aggregation technique, we define the one step transition probability and cost for newly defined aggregate states $S_i, S_j \subseteq \mathbf{S}$ and $d \in \mathcal{D}_{S_i}$ as follows, respectively.

$$P(S_j|S_i, d) = \frac{1}{|S_i|} \sum_{s \in S_i} \sum_{s' \in S_j} P(s'|s, d) \tag{4.11}$$

$$c(S_i, d) = \frac{1}{|S_i|} \sum_{s \in S_i} c(s, d) \tag{4.12}$$

It is easy to see that 4.11 are the transition probabilities in the aggregate smaller MDP.

$$\begin{aligned} \sum_{i \in N} P(S_j|S_i, d) &= \sum_{i \in N} \frac{1}{|S_i|} \sum_{s \in S_i} \sum_{s' \in S_j} P(s'|s, d) \\ &= \sum_{s \in S_i} \frac{1}{|S_i|} \sum_{i \in N} \sum_{s' \in S_j} P(s'|s, d) \\ &= \sum_{s \in S_i} \frac{1}{|S_i|} \sum_{s' \in S} P(s'|s, d) = 1 \end{aligned} \tag{4.13}$$

The output of the aggregation procedure is certainly an MDP, but one with fewer states. This enables us to use LP to solve the aggregate model as well. After solving the small MDP with LP, we obtain the approximate optimal policy for the aggregate model. Note that this policy can also be implemented for the original MDP since the set of aggregate actions is inherited from the original states for each aggregate meta-state. Obviously, the optimal decisions can not exactly match for both the original and the aggregate MDP, but we can evaluate the approximate optimal policy on the original model. Assume $\bar{\pi}$ is the approximate optimal policy, a feasible policy, derived form $\bar{\pi}$, on the original MDP δ is evaluated as below.

$$\delta(s) = d \text{ if } s \in S_n \text{ and } \bar{\pi}(S_n) = d \ \ \forall n \in N \tag{4.14}$$

The evaluation in (4.14) indicates the use of the same optimal action in states that are assigned to the same aggregate meta-state ($s \in S_k$).

CHAPTER 5

COMPUTATIONAL RESULTS

In this chapter, we present and discuss our computational observations. Our solution methods use the structural results to identify suboptimal policies, and offer higher efficiency to find optimal policies. We also consider state space reduction techniques to be able to solve larger instances which would not be possible otherwise. The methods using the reduction techniques offer a trade off between the solution quality and the computation time. We perform extensive computational studies to observe the practical performance of our solution methods and also compare them with each other and the basic LP formulation. In Table 5.1 we summarize our solution methods as a reminder, stating their main features.

Table 5.1: Solution Methods

Solution Method	Features
Basic LP Formulation	Basic LP provides long-run average cost optimal policies.
Reduced LP formulation	Structural results eliminate suboptimal actions and reduced LP handled. Suboptimal actions do not affect the optimal decision: RLP hits the same optimal cost as BLP.
State space decomposition	All transient states under all deterministic policies are identified and removed from $\mathcal{S}$ A smaller MDP model is generated, then a smaller LP is provided. The smaller LP is solved optimally and the same optimal cost found as BLP.
State space decomposition & structural results	Structural results integrate into method in different two steps: (i) before, and (ii) after the decomposition of $\mathcal{S}$.
State aggregation	State aggregation methods are used to generate aggregate MDP models. Optimal solutions of aggregate MDP models provide approximate optimal solutions to the original MDP model.
State aggregation & structural results	Integrating structural results into aggregation method, different aggregate models are generated.

Following the order in Table 5.1, we first illustrate the effect of the proposed structural results on the performance of the basic LP formulation. We compare the results obtained by the basic LP (BLP) with our reduced LP (RLP). We evaluate the contribution of structural results in terms of solution quality, computation time and memory requirement on small and medium size instances. Basic LP cannot find the optimal solution on large instances, since the LP solver is incapable to generate associated LP due to physical memory constraints, we also work on approximation techniques. Therefore, we compare our results with to benchmark heuristics on those instances. The first benchmark heuristic is a myopic policy (MP) that picks the minimum-cost action at each state. The second benchmark policy, which we call always-serve policy (AP), forces the system to work in full server capacity and aims to serve as many jobs as possible. To enhance the quality of solutions and the computation times of these

heuristics, we first implement action elimination.

5.1 Generation of Experimental Scenarios

We generate our experimental scenarios by varying length of the customer order horizon K, number of servers M, maximum number of arrivals for a time period A, and arrival rate λ. We also experiment on different customer order and priority patterns. We refer to customers' order pattern as the *load* of the system. We use the term *segmentation* to describe the distribution of arriving customers' priorities. Segmentation and load are integrated into our formulation through q_i and v_j probabilities, respectively. We formally define four different segmentation patterns, which we use to compute different q_i probabilities below.

1. *Equally segmented systems* (ES): Customers arriving at time t are equally likely to be high-priority or low-priority, $q_i = \frac{1}{2},\ i = 1, 2$.

2. *High-priority systems* (HS): Customers arriving at time t are more likely to be high-priority.
$$q_i = \frac{(3-i)^2}{\sum_{i=0}^{K} i^2},\ i = 1, 2$$

3. *Low-priority systems* (LS): Customers arriving at time t are more likely to be low-priority.
$$q_i = \frac{i^2}{\sum_{i=0}^{K} i^2},\ i = 1, 2$$

4. *Arbitrary segmented systems* (AS): Customers requests are randomly distributed over the planning horizon. q_i's are random numbers $\sum_{i=1}^{2} q_i = 1$ where and $q_i > 0$.

For the high-priority and low-priority systems, we use quadratic functions to model the change in q_i 's respect to i. However, the in change in q_i 's can be captured with different functions. Similarly we define four different load patterns, which we use to compute different v_j probabilities below.

1. *Equal load systems* (EL): Customers arriving at time t are equally likely to request a service for any $t + j$, where $0 \leq j < K$.

$$v_j = \frac{1}{K}, \ \forall j$$

2. *Front-loaded systems* (FL): Customers arriving at time t are more likely to request a service for $t + j_1$ than $t + j_2$ whenever $j_1 < j_2$.

$$v_j = \frac{(K-j)^2}{\sum_{i=0}^{K} i^2}, \ \forall j$$

3. *Back-loaded systems* (BL): Customers arriving at time t are more likely to request a service for $t + j_1$ than $t + j_2$ whenever $j_1 > j_2$.

$$v_j = \frac{(j+1)^2}{\sum_{i=0}^{K} i^2}, \ \forall j$$

4. *Arbitrary-loaded systems* (AL): Customer requests are randomly distributed over the planning horizon. v_j's are random numbers $\sum_{j=0}^{K-1} v_j = 1$ where and $v_j > 0$.

For the front-loaded and the back-loaded systems, we assume a quadratic change in v_j 's respect to j. Corresponding v_j 's can be computed easily if the load over the planning horizon is changing at a different rate.

To simulate discrete customer arrivals, we use truncated Poisson distribution. Different segmentation and loads in the system can be captured by using different rate λ_{ij} for the Poisson arrivals for each customer class i and each period j. For an overall arrival rate λ, the arrival rate of a i-priority class job request for time $t + j$ is λ_{ij}. Once the q_i and v_j are computed, $\lambda_{i,j}$ is computed as follows: $\lambda_{ij} = q_i v_j \lambda$. Then, the truncated Poisson arrival probabilities can be computed according to following equation.

$$p_{ij}(a) = \frac{\frac{e^{\lambda_{ij}} \lambda_{ij}{}^a}{a!}}{\sum_{n=0}^{A} \frac{e^{\lambda_{ij}} \lambda_{ij}{}^n}{n!}}, \text{ for } i = 1, 2 \text{ and } 0 \leq j < K \tag{5.1}$$

We generate parameters for experiments by crossing different values of the problem parameters: number of servers M, general arrival rate λ, maximum number of arriving job requests for $t + j$ A, length of the planning horizon K, the cost ratios c_o/c_r,

Table 5.2: Instance Parameters

M	$\in$	$\{1, 2, 5\}$
A	$\in$	$\{1, 2, 3, 4, 5, 6\}$
λ	$=$	$0.2A$
K	$\in$	$\{2, 3, 4\}$
c_o	$=$	2000
c_r	$\in$	$\{500, 1000, 1500\}$
c_1^e	$\in$	$\{1000, 1500, 3000\}$
c_2^e	$\in$	$\{500, 1500, 2500\}$

c_o/c_2^e, c_o/c_1^e, the segmentation and the load of the system. The values we used in our experiments are given in the table below.

Some of these parameter combinations result in very large state spaces that exceed our computational power. Therefore, such combinations are omitted. All models are coded in C++ programming language with IBM ILOCPLEX Concert Technology, and all experimental scenarios are tested by Visual Studio 19 based on Intel(R) Core(TM) i7 Processor 3.10 GHz with 16.0 GB RAM under the 64-bit Microsoft Windows 10 operating system. Benchmark heuristics and approximate model minimization methods are also coded in C++ programming language and Armadillo C++ [34] library is used for internal matrix operations.

5.2 Computational Results

In this subsection, we present our numeric computational results through comparative tables. Thus, we use the same notations in each table unless otherwise stated. In the tables presented below, K, M, A, c_o, c_r, c_1^e, c_2^e are the problem parameters as described earlier. $|\mathcal{S}|$ and $|\mathcal{D}|$ denote the number of states and the size of the action space, respectively. Cost is the long-run average cost. $t(sec)$ is the total computation time in seconds. M(MB) is the total memory used during computations in MB.

Before moving on to our computational experiments, in Table 5.3, we first show the

MDP models' sizes that we can solve optimally on the original MDP, to give an insight about the contributions of our action space reduction method, structural results. In Table 5.3, the very first row indicates the three different cost structures: $C1 : c_2^e < c_1^e < c_r < c_o$, $C2 : c_r < c_2^e < c_1^e < c_o$, $C3 : c_r < c_o < c_2^e < c_1^e$. Remember that the number of states determines the number of rows in the LP formulations, and the number of state-action pairs determine the number of columns. The columns under M values show the number of decision variables for BLP and RLP, respectively.

The success of revealed structural results is significant for scarce and moderate server capacities. For instance, $M = 1$ is scarce for all scenarios, since $1 \leq A$, and in each scenario the number of decision variables decreases to approximately one third. As the server capacity increases structural results lose their effectiveness, e.g. when $K = 2$, $M = 5$ and $A = 1$ under some cost structures structural results can not identify any suboptimal actions, since it is possible to serve all request on their preferred service completion times at a minimum cost with these parameters. However, when systems under $C3$ are considered, since early service is found to be suboptimal, independent of the server capacity, structural results ensures elimination of such actions.

Table 5.3: BLP and RLP dimensions under different cost structures

						$C1$			$C2$			$C3$		
			BLP			RLP								
			$M=1$	$M=2$	$M=5$	$M=1$	$M=2$	$M=5$	$M=1$	$M=2$	$M=5$	$M=1$	$M=2$	$M=5$
K	A	$\lvert\mathcal{S}\rvert$	Number of decision variables											
	1	48	118	145	215	77	127	215	71	127	215	65	90	108
	2	405	1683	1896	3646	589	1040	3430	538	864	3141	517	728	1404
2	3	1792	11416	12210	21374	2421	4076	17295	2237	3262	14153	2185	2952	7020
	4	5625	51175	53290	81760	7226	11546	53725	6746	9238	37110	6641	8640	21450
	5	14256	175806	180435	248032	17665	26969	126252	16627	21780	74484	16441	20750	48776
3	1	1280	4962	7169	20925	1977	4218	15132	1977	4217	15132	1887	3492	7465

Similarly, Table 5.4 shows the size of original and aggregate MDP models, generated through total job aggregation method, to illustrate the contributions of our problem-specific state space reduction method. These instances can not handle optimally, due to memory constraints. Although the total work aggregation is a very primitive method that uses structural properties, it has reduced the number of states significantly and enabled these problems to be solved with LP. However, when used with action

elimination, it creates a big difference in all spaces of the problem.

Table 5.4: Original and aggregate MDP dimensions under an arbitrary cost structure

		Original MDP				Aggregate MDP			
			$M=1$	$M=2$	$M=5$		$M=1$	$M=2$	$M=5$
K	A	$\lvert\mathcal{S}\rvert$	Number of decision variables			$\lvert\mathcal{S}\rvert$	Number of decision variables		
2	6	31213	501613	510510	647755	6517	106477	113169	176535
3	2	59049	496224	593211	1845335	7425	52774	67432	166939
4	1	64512	376858	569197	2282873	10152	35451	37704	82599

According to Tables 5.3 and 5.4, among all parameters, the length of the planning horizon K emerges as the most influential problem parameter on the dimensionality of the problem. As the length of the planing horizon increases, the size of the state space increases exponentially. This increase certainly imposes a computational challenge which results in longer computation time and larger memory requirement for the LP.

The factors that determine the performance of the LP are the length of the customer order horizon, the maximum number of arriving job requests for $t + j$, the server capacity, the ratios between costs, the segmentation and the load of the system. Therefore, our discussions revolve around these parameters.

5.2.1 Action Space Reduction: Structural Results

The structural results provides some conditions to determine a set of suboptimal actions. These conditions are dependent on problem parameters such as costs, server capacity and length of the customer order horizon. Therefore, we put these parameters in the center of our discussion under this subsection, then we also emphasize about the effect of other parameters.

We first evaluate the results in Table 5.5, in order to clarify our segmentation and load choices over discussions, given in the following results. To create this table, we compute the solutions of many different scenarios with different cost and arrival probabilities and we come up with the following 5.5 as a summary. In Table 5.5,

we present the results of a system with $K = 2$, $A = 4$ and $M = 2$, under different segmentation and load patterns, when $c_o = 2000$, $c_r = 500$, $c_1^e = 3000$ and $c_2^e = 2500$. Observing the computation time of BLP with different arrival probabilities, we see that the most challenging formulations appear when the systems are equally segmented. Among the loads, we find back-loaded systems as easily solvable compare to others. Therefore, in the following tables, we give results for the systems which are equally segmented.

Table 5.5: Computational results on $K = 2$, $M = 2$, $A = 4$, $c_o = 2000$, $c_r = 500$, $c_1^e = 3000$ and $c_2^e = 2500$

			BLP	RLP				BLP	RLP
Segment	Load	Cost	t(sec)	t(sec)	Segment	Load	Cost	t(sec)	t(sec)
ES	EL	58.05	311.72	49.28	LS	EL	53.75	252.60	54.47
	FL	717.23	354.56	65.21		FL	697.70	307.16	89.76
	BL	0.32	137.43	28.74		BL	0.31	131.63	28.06
	AL	104.54	177.20	52.21		AL	71.05	222.19	59.53
HS	EL	48.69	226.91	52.58	AL	EL	41.69	151.61	63.71
	FL	658.78	288.93	85.27		FL	307.39	182.81	62.45
	BL	0.31	130.48	28.57		BL	0.13	133.58	28.04
	AL	35.00	217.67	46.28		AL	12.05	186.30	32.46

To observe the behavior of structural results under different cost structures, in Table 5.6, we present the results corresponding to an equally segmented and equally loaded system where $K = 2$ and A varies from 3 to 5, under different cost structures: $C1 : c_2^e = 500 < c_1^e = 1000 < c_r = 1500 < c_o = 2000$, $C2 : c_r = 500 < c_2^e = 1000 < c_1^e = 1500 < c_o = 2000$, $C3 : c_r = 1000 < c_o = 2000 < c_2^e = 2500 < c_1^e = 3000$. We set the service capacity to 2, since $M = 2$ can be considered as moderate for most of the scenarios, according to Table 5.3.

The number of states and actions increases with A for each LP formulation, such that basic and reduced. Apart from cost structure, this is an other parameter that directly affect the performance of LP and will be discussed later. However, now observe that, $|\mathcal{D}|$ is same for each A in the basic LP formulation under different costs. For the reduced formulation, in each experiment, we observe significant reduction in the number of decision variables, hence the number of total allowable actions. That is the policy space, where the simplex algorithm tries to find out some basic feasible

Table 5.6: Computational results on $K = 2$, $M = 2$ under different cost structures

				BLP			RLP		
	A	$\|\mathcal{S}\|$	Cost	$\|\mathcal{D}\|$	$t(sec)$	M(MB)	$\|\mathcal{D}\|$	$t(sec)$	M(MB)
	3	1792	41.21	12210	15.45	506.78	4076	6.22	169.62
$C1$	4	5625	86.87	53290	293.05	6914.27	11546	86.68	1499.74
	5	14256	152.13	180435	-	-	26969	1037.70	8863.18
	3	1792	26.26	12210	13.67	506.78	3262	5.14	135.89
$C2$	4	5625	57.57	53290	269.06	6914.27	9238	55.33	1200.39
	5	14256	101.61	180435	-	-	21780	478.81	7158.91
	3	1792	35.47	12210	14.69	506.78	2952	4.51	123.04
$C3$	4	5625	77.42	53290	274.93	6914.27	8640	49.29	1122.83
	5	14256	138.61	180435	-	-	20750	523.22	6820.61

solutions, gets narrower. The resulting reduction in the number of columns in the LP formulation, leads significant improvements in computation times and memory requirements. For instance, in each problem, the number of decision variables reduces near to one forth. Overall, in each instance running time halves and memory requirement reduces near to one forth. Further, observe that in Table 5.6 for systems with $A = 5$, computation time and memory requirements of BLP are marked with "-" since these instances are beyond our computational power. Once suboptimal actions removed, dimension of the transition probability matrix reduces for these instances, results show that the number of columns in the LP formulation reduces by 75% in our RLP formulation. Then, we are able to solve them optimally through RLP. We observe a similar enhancement under different costs for systems with $A = 6$ and $M = 1$, but this is different in the mean of contribution of structural results, since $M = 2$ is moderate and $M = 1$ is scarce.

For systems with $K = 2$, the effect of action elimination is obvious respect to different cost structures. For systems with longer customer order horizon, e.g. $K > 2$, the generalized forms of action elimination conditions can be applicable. However, as it is mentioned, in these cases decisions become more inter-dependent and the power of action elimination is not that much evident. Our computational power incapacitates when $K = 4$, thus we provide only computational results on $K = 3$ in the following table. Table 5.7 shows the results for equally segmented and equally loaded systems with $A = 1$ and $M = 1, 2, 5$, under different cost structures, e.g. $C1$, $C2$ and $C3$.

Table 5.7: Computational results on $K = 3$, $A = 1$ under different cost structures

			BLP			RLP		
	M	Cost	$\|\mathcal{D}\|$	t(sec)	M(MB)	$\|D\|$	t(sec)	M(MB)
	1	20.09	4962	4.78	147.66	2098	2.48	1.88
$C1$	2	0.93	7169	5.80	213.17	5451	4.59	162.17
	5	0.00	20925	13.42	621.22	20781	13.01	616.95
	1	15.23	4962	4.52	147.66	1977	1.60	59.15
$C2$	2	0.63	7169	5.93	213.17	4217	3.60	125.67
	5	0.00	20925	13.15	621.22	15147	9.90	10.15
	1	16.50	4962	3.85	147.66	1887	1.50	56.48
$C3$	2	0.66	7169	5.15	213.17	3487	2.77	103.93
	5	0.00	20925	13.51	621.22	7478	5.99	222.37

As we expected, running times increase for both formulations, since the length of customer order horizon increases. We notice approximately 75% reduction in running times of each instance. However, improvements in memory requirements are not that much significant, since the impact of structural results weaken with increasing K values.

The number of servers in the system also directly increases the complexity of the problem. As shown in Table 5.7, while the number of states in the state space depends on A for a fixed length of customer order horizon, the number of the actions that are valid in each state varies depending on the server capacity of the system. As the capacity of the system increases, the number of requests rejected, and the number of requests served in overtime decreases. Moreover, when the server capacity is large enough, the trade-off between rejection and early service, or the trade-off between rejection and do not serve decisions gains importance. For each test instance, there exist a limit for the server of capacity, from which the capacity widens, each service request can be delivered to the lowest cost at any time. Note that, in such cases the structural results do not contribute to the solution, since all actions are allowable. In short, as M increases, the ability of action elimination decreases, since in systems with abundant server capacity there is no restriction on decisions and all feasible policies are candidates to be optimal.

In Table 5.8, we present the results corresponding to an equally segmented system where $K = 2$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$, to show the contribution of structural results given in Proposition 1. A varies from 3 to 5. We also give results under different system loads. However, $|\mathcal{D}|$ is same for each A in RLP. Remember that by definition structural results are based on only problem parameters M and costs. Thus, we identify the same actions as suboptimal under different loads.

Table 5.8: Computational results on $K = 2$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$

				BLP			RLP		
	A	$\|\mathcal{S}\|$	Cost	$\|\mathcal{D}\|$	t(sec)	M(MB)	$\|D\|$	t(sec)	M(MB)
	3	1792	41.21	12210	15.28	506.78	4076	6.18	169.60
EL	4	5625	86.87	53290	293.55	6914.27	11546	86.68	1499.66
	5	14256	152.13	180435	-	-	26969	1037.70	8862.88
	3	1792	41.56	12210	18.89	506.78	4076	7.91	169.60
FL	4	5625	91.09	53290	289.99	6914.27	11546	112.96	1499.66
	5	14256	162.89	180435	-	-	26969	886.88	8862.88
	3	1792	31.39	12210	16.66	506.78	4076	8.68	169.60
BL	4	5625	71.33	53290	290.51	6914.27	11546	120.40	1499.66
	5	14256	132.02	180435	-	-	26969	983.22	8862.88
	3	1792	3.40	12210	11.02	506.78	4076	4.35	169.60
AL	4	5625	72.93	53290	204.17	6914.27	11546	62.38	1499.66
	5	14256	132.84	180435	-	-	26969	538.32	8862.88

The number of states increases with A and the effect of Proposition 1 becomes more relevant. In each experiment, we observe significant differences in computation time and in memory requirements. For instance, in each problem, the number of decision variables reduces near to one forth. This reduction follows by significant decreases in the computation times and BLP's memory requirement. Although the structural results are independent of the arrival probabilities, we observe that the decrease in computation time in the arbitrarily loaded systems is more significant than in the others. The main reason for this is that the complexity of LP itself is also affected by arrival probabilities. When the transition probability matrix is sparse due to arrival probabilities, the LP formulation itself become more simple, and trough action elimination, basic LP is probably reduced to a much simpler formulation.

Results presented in Tables 5.9 and 5.10 show the impact of Proposition 2 and Proposition 3. These tables also present the results for distinct system loads since the probabilities of arrival only affect the optimal cost, not the functioning of structural results.

Table 5.9: Computational results on $K = 2$, $M = 2$, $c_o = 2000$, $c_r = 500$, $c_1^e = 1500$ and $c_2^e = 1000$

				BLP			RLP		
	A	$\|\mathcal{S}\|$	Cost	$\|\mathcal{D}\|$	t(sec)	M(MB)	$\|\mathcal{D}\|$	t(sec)	M(MB)
	3	1792	26.26	12210	15.64	506.78	3262	4.54	135.89
EL	4	5625	57.57	53290	273.52	6914.27	9238	53.42	1200.39
	5	14256	101.61	180435	-	-	21780	356.64	7158.91
	3	1792	22.16	12210	16.89	506.78	3262	5.28	135.89
FL	4	5625	48.65	53290	235.68	6914.27	9238	50.63	1200.39
	5	14256	87.04	180435	-	-	21780	363.68	7158.91
	3	1792	23.48	12210	15.45	506.78	3262	5.09	135.89
BL	4	5625	51.95	53290	225.20	6914.27	9238	75.56	1200.39
	5	14256	93.27	180435	-	-	21780	574.83	7158.91
	3	1792	18.08	12210	13.16	506.78	3262	4.91	135.89
AL	4	5625	42.07	53290	165.46	6914.27	9238	33.21	1200.39
	5	14256	82.34	180435	-	-	21780	444.67	7158.91

We observe that among the three propositions, Proposition 3 is the most effective since in systems with greater early service costs, the problem of finding the optimal admission control and capacity allocation reduces into a more simple admission control problem with overtime costs.

Table 5.10: Computational results on $K = 2$, $M = 2$, $c_o = 2000$, $c_r = 1000$, $c_1^e = 3000$ and $c_2^e = 2500$

				BLP			RLP		
	A	$\|\mathcal{S}\|$	Cost	$\|\mathcal{D}\|$	t(sec)	M(MB)	$\|D\|$	t(sec)	M(MB)
	3	1792	26.48	12210	15.11	506.78	2952	3.71	123.04
EL	4	5625	58.04	53290	282.89	6914.27	8640	41.62	1122.83
	5	14256	102.47	180435	-	-	20750	383.97	6820.61
	3	1792	22.18	12210	15.33	506.78	2952	4.66	123.04
FL	4	5625	48.69	53290	279.06	6914.27	8640	55.14	1122.83
	5	14256	87.10	180435	-	-	20750	411.78	6820.61
	3	1792	24.28	12210	15.46	506.78	2952	4.15	123.04
BL	4	5625	53.75	53290	258.98	6914.27	8640	56.49	1122.83
	5	14256	96.32	180435	-	-	20750	314.38	6820.61
	3	1792	13.93	12210	14.25	506.78	2952	3.82	123.04
AL	4	5625	34.29	53290	199.29	6914.27	8640	32.02	1122.83
	5	14256	62.73	180435	-	-	20750	445.16	6820.61

Action elimination procedure, based on our propositions, is a well performing off-line iterative search, which occurs in the entire action space. Then, we naturally expect that with its implementation, the preprocessing time, which includes generation of MDP elements and linear programming element, increases. In contrast, it sees a decrease due to the narrower feasible region, since the most time-consuming procedure is the generation of coefficients of constraints. Additionally, we see a significant decline in the resolution time of the simplex algorithm. For instance, this fact can be seen in Table 5.12, showing the detailed time measurements for an equally segmented and equally loaded system.

Table 5.11: Computational performance of BLP and RLP when $K = 2$, $M = 2$, $c_o = 2000$, $c_r = 500$, $c_1^e = 1500$ and $c_2^e = 1000$

	BLP			RLP		
A	$\|\mathcal{D}\|$	Setup t(sec)	Algorithm t(sec)	$\|\mathcal{D}\|$	Setup t(sec)	Algorithm t(sec)
1	145	0.01	0.04	127	0.00	0.03
2	1896	0.11	0.39	864	0.01	0.16
3	12210	0.73	5.11	3262	0.05	1.17
4	53290	3.39	109.17	9238	0.36	20.54

So far, we discussed the effect of the structural results on LP formulation on tractable instances. To summarize briefly, when suboptimal decisions are removed from the action space $\mathcal{D}$ in our reduced LP model, the memory requirement of the LP decreases considerably and the problems that are first intractable can be solved optimally. On the other hand, we see significant improvement in computation time for each instance. Since LP finds the optimal deterministic and stationary capacity allocation policy, once an optimal policy determined, the system controller can use the result until arrival probabilities change.

5.2.2 State Space Reduction

The second part of our computational experiments focus on larger instances where we employ our aggregation methods. Through aggregation, we decrease the size of the state space, and can solve larger instances. As mentioned in Table 5.1 we follow two approaches to reduce the size of the state space: (i) the decomposition of the state space, (ii) state aggregation methods.

5.2.2.1 Decomposition of State Space

To decompose the state space, we use the decomposition algorithm in [11] to determine the set of states that form the recurrent set under all deterministic policies. Then we reformulate the LP on this reduced and equivalent state space. To solve the MDP model, we use the following steps: (i) restrict the MDP to recurrent class, (ii) remove suboptimal actions from the reduce action space, and (iii) reformulate the associated LP, thus solve it with an appropriate algorithm. We present the related computational results in the following.

Table 5.12 shows results for an equally loaded systems where $K = 2$, $M = 2$ and A varies from 1 to 3. We observe the system under different segmentation patterns since decomposition algorithm uses accessibility between states which is mainly determined by arrival probabilities. These results represent the effect of the decomposition algorithm on the performance of the LP, and gives insight about the limiting structure of the MDP. In the tables below, addition to already existing columns we add

$dect(sec)$ which shows the the computation time of the decomposition in seconds.

Table 5.12: Computational results on equally systems $K = 2$, $M = 2$ $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$

				Without Decomposition						With Decomposition							
				BLP			RLP					BLP			RLP		
	A	Cost	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{D}\rvert$	t(sec)	M(MB)	$\lvert D\rvert$	t(sec)	M(MB)	$\lvert\mathcal{S}\rvert$	dect(sec)	$\lvert\mathcal{D}\rvert$	t(sec)	M(MB)	$\lvert D\rvert$	t(sec)	M(MB)
	1	0.69	48	145	0.04	0.27	127	0.03	0.25	48	0.00	145	0.04	0.27	127	0.04	0.25
ES	2	12.36	405	1896	0.75	18.23	1040	0.47	10.09	405	0.30	1896	1.03	18.23	1040	0.77	10.09
	3	41.21	1792	12210	14.44	506.78	4076	6.57	169.60	1792	10.04	12210	24.35	506.78	4076	16.46	169.60
	1	8.16	48	145	0.04	0.27	127	0.04	0.25	48	0.01	145	0.05	0.27	127	0.04	0.25
HS	2	145.87	405	1896	0.72	18.23	1040	0.40	10.09	405	0.34	1896	1.03	18.23	1040	0.74	10.09
	3	472.28	1792	12210	19.24	506.78	4076	7.49	169.60	1792	10.00	12210	27.90	506.78	4076	17.49	169.60
	1	0.00	48	145	0.01	0.27	127	0.01	0.25	8	0.02	23	0.03	0.08	23	0.03	0.08
LS	2	0.09	405	1896	0.44	18.23	1040	0.26	10.09	33	1.35	153	1.38	0.23	153	1.30	0.23
	3	0.37	1792	12210	9.92	506.78	4076	3.86	169.60	90	51.98	569	52.07	1.38	569	52.23	1.38

Now consider the solutions for system where the maximum number of arrivals is $A = 3$; as expected, the number of actions is quite huge. Therefore, in the case of the full state space, the solution of the linear program takes long time. Our propositions, without the implementation of the decomposition algorithm, are more successful in terms of computation time. This large time difference between these two well-functioning approaches originates from the depth-first search performed for each state during the decomposition algorithm. Again, when we examine the results in Table 5.10, the solution of the linear program takes a short time, with or without the action elimination procedure, after the decomposition of the state space. However, almost all of the computation time is due to the processing time of the decomposition algorithm. These results show that structural results is more efficient in enhancement of the LP's performance.

In Chapter 3, we showed that the MDP is unichain. Depending on the arrival probabilities, the state space of the MDP consists of a single recurrent class and a set of possibly empty transient states. We present the related results in Table 5.12 For some arrival probabilities, for example in equally segmented and high-priority segmented systems, the model has an empty set of transient states. In such cases, the decomposition algorithm can not detect any. Therefore, for these scenarios, the decomposition algorithm only increases the computation time. We conclude that our structural results can be implemented to general settings, as they operate independently of the

arrival possibilities; in contrast, the decomposition algorithm is not always beneficial.

Table 5.12 shows that, for appropriate scenarios, the implementation of the decomposition algorithm dramatically reduces the number of states, to be considered in LP. This noticeable reduction in the state space stimulates a considerable decrease in the memory requirement of the solution algorithm. Further, when we use the structural results following the decomposition algorithm, the action space then becomes even narrower. However, regarding the computation times, the time of the problem reduced with the algorithm does not seem to differ from the time of the basic LP significantly as we mentioned above. Therefore, a trade-off appears between the problem scale and the computation time. From a practical view, despite its enormous computation requirement, the decomposition algorithm allows us to find the optimal policies of the problems that are still intractable with LP, even after the action elimination.

As shown in Chapter 3, the MDP model is unichain; almost all nodes, all states, are connected in the G_2 graph. This fact significantly increases the duration of the search algorithm used to find strongly connected components. Therefore, we observe that we can remove suboptimal actions before applying the algorithm to reduce the required time to decompose the the state space. Table 5.13 shows the effect of elimination of suboptimal actions before the decomposition of the state space for low-priority segmented systems where $M = 2$. The removal of suboptimal actions from the action space is equivalent to the removal of some edges from the G_2 graph.

Table 5.13: Computational results on $M = 5$, $K = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$

				Without Decomposition						With Decomposition				
				BLP			RLP			RLP				
	A	Cost	$\|S\|$	$\|D\|$	t(sec)	M(MB)	$\|D\|$	t(sec)	M(MB)	$\|S\|$	dect(sec)	$\|D\|$	t(sec)	M(MB)
	1	0.46	48	145	0.04	0.27	127	0.03	0.25	48	0.01	127	0.04	0.25
EL	2	8.70	405	1896	0.86	18.23	1040	0.45	10.09	405	0.19	1040	0.63	10.09
	3	31.39	1792	12210	21.09	506.78	4076	7.56	169.60	1792	4.54	4076	12.94	169.60
	1	5.48	48	145	0.07	0.27	127	0.07	0.25	48	0.01	127	0.08	0.25
FL	2	108.40	405	1896	0.75	18.23	1040	0.47	10.09	405	0.22	1040	0.69	10.09
	3	399.60	1792	12210	19.08	506.78	4076	8.76	169.60	1792	4.40	4076	14.00	169.60
	1	0.00	48	145	0.02	0.27	127	0.01	0.25	8	0.01	23	0.02	0.08
BL	2	0.06	405	1896	0.42	18.23	1040	0.26	10.09	72	0.70	156	0.73	0.38
	3	0.26	1792	12210	9.93	506.78	4076	4.01	169.60	388	18.34	629	18.54	5.93

From the results of Tables 5.12 and 5.13, we can deduce that the decomposition algorithm is successful for systems with shorter rolling planning horizons and lower arriving rates since it provides an optimal solution in an acceptable time period. However, even for these problem settings, our propositions contribute more in terms of computation time. This phenomenon is inherited from the depth search employed for each node on the G_2 graph. For larger problems, apart from the MDP, the search on G_2 itself becomes intractable because of the immense memory requirement when the customer order horizon gets wider.

5.2.2.2 State Aggregation and Aggregate MDP Model

We perform state aggregation to reduce the size of the state space and the action space, to make the larger problems tractable. Once we generate the aggregate MDP model, we solve it optimally via LP. The solution to the aggregate MDP, is an approximate optimal solution for the original MDP model. When the server capacity is scarce the accuracy of the aggregate MDP model is high, however under some conditions the approximate optimal policy can be very different from the optimal policy. In order to asses the quality of the aggregate MDP, in optimally solvable instances we use two performance indicators: (i) absolute gap (AG) and (ii) matched action percentage (MAP).

$$\text{AG} = 100 \times \frac{\text{Approximate optimal average cost - Optimal average cost}}{\text{Optimal average cost}} \tag{5.2}$$

$$\text{MAP} = 100 \times \frac{\text{Number of same actions across all states in } S}{\text{Total number of states } (|S|)} \tag{5.3}$$

AG is an obvious indicator, and itself it demonstrates the percent deviation from the optimal long-run average cost. MAP shows the average accuracy of the aggregate MDP model in terms of its ability to find the optimal decision across all states of the original MDP model.

For larger instances, we compare the aggregate MDP model's performance with two benchmark heuristic policies, MP and AS that we introduce formerly.

To ensure the ease of implementation, in our computations we set c_r and c_p positive coefficients introduced in Definition 4 as $c_r = \max_{s \in \mathcal{S}} c(s, d)$ and $c_p = 1$, so that the ε can take values in $[0, 1]$. Therefore, for $\varepsilon = 0$, if our MDP does not have equivalent states as defined in 3, in terms of costs and transition probabilities, we have the full MDP. Fortunately, our MDP model has many equivalent states, thus we can generate more accurate approximate optimal policies. Certainly, for ε values closer to 1, the aggregate MDP model does not completely reflect our system considerations, especially early service decisions.

Table 5.15 shows the computational results on an equally segmented and equally loaded system when $K = 2$, $A = 4$ and $M = 2$. In the following tables, the optimal solution is indicated as OPT and solution of each ε-aggregate MDP model is indicated as ε-agg. In addition to already existing columns we add $aggt(sec)$, MAP and AG columns. These columns shows the computation time of aggregation heuristic in seconds, MAP in percentage and AG in percentage, respectively. We measure the accuracy of each ε-aggregate MDP model respect to AG and MAP. We also show the running time of the aggregation method to observe the trade-off between solution quality and solution time throughout the changes in ε.

Table 5.14: Computational results on $K = 2$, $A = 4$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$

	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{D}\rvert$	Cost	M(MB)	t(sec)	aggt(sec)	MAP(%)	AG(%)
OPT	5625	53290	86.87	6914.27	273.65	-	-	-
0-agg	251	1509	99.54	13.86	61.90	1.81	64.27	14.58
0.2-agg	67	206	100.09	1.02	47.04	0.47	63.68	15.21
0.4-agg	66	198	100.13	1.00	46.69	0.46	61.56	15.26
0.6-agg	65	195	100.59	0.99	47.06	0.46	62.54	15.78
0.8-agg	65	195	100.59	0.99	46.94	0.45	66.35	15.78
1-agg	65	195	100.59	0.99	46.33	0.07	62.84	15.78

As ε increases, the number of states significantly decreases. However, from $\varepsilon = 0.6$, neither the number of aggregate states nor the number of decision variables changes. This fact suggests that not only the ε but also the costs and arrival probabilities are important for the ε-aggregation method. In Table 5.15 and Table 5.16 we present results for the same system parameters except we observe the system under different

system loads, e.g. FL and BL, respectively.

Table 5.15: Computational results on $K = 2$, $A = 4$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$, under front-loading

	$\|\mathcal{S}\|$	$\|\mathcal{D}\|$	Cost	M(MB)	t(sec)	aggt(sec)	MAP(%)	AG(%)
OPT	5625	53290				-	-	-
0-agg	262	1496	94.16	14.27	60.70	1.52	73.46	3.37
0.2-agg	68	220	94.20	1.06	45.92	0.44	73.24	3.41
0.4-agg	67	206	94.20	1.02	46.00	0.45	74.54	3.41
0.6-agg	65	195	94.23	0.99	47.22	2.25	72.02	3.44
0.8-agg	65	195	94.23	0.99	46.00	0.44	73.80	3.44
1-agg	65	195	94.23	0.99	45.74	0.07	72.75	3.44

Observing the same system under different loads, that is, with different arrival probabilities, we see that the accuracy of the aggregate MDP models is not only dependent on the value of ε. Therefore, when applying the ε-aggregation method, the chosen arrival probabilities should be compatible with the aggregation rule, such that it can able to reflect the original MDP's considerations.

Table 5.16: Computational results on $K = 2$, $A = 4$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$, under back-loading

	$\|\mathcal{S}\|$	$\|\mathcal{D}\|$	Cost	M(MB)	t(sec)	aggt(sec)	MAP(%)	AG (%)
OPT	5625	53290				-	-	-
0-agg	260	1379	103.87	13.20	60.20	1.62	67.41	45.61
0.2-agg	67	206	107.00	1.02	48.76	0.44	66.33	50.01
0.4-agg	66	198	107.27	1.00	46.79	0.45	65.74	50.38
0.6-agg	65	195	108.87	0.99	46.13	0.44	68.34	52.63
0.8-agg	65	195	108.87	0.99	48.19	0.44	66.52	52.63
1-agg	65	195	108.87	0.99	46.80	0.06	67.66	52.63

The cost for the aggregated MDP models is computed through the approximate optimal policies. The approximate optimal policy is evaluated according to relation 4.14, and a feasible policy for the original MDP is obtained. The approximate optimal cost increases respect to the increase in ε value. However, the results in the three above tables show that the increase in optimal cost, that is worsening of solution quality, can be considered as insignificant compared to the improvement in computation time.

Therefore, for intractable instances we only provide the results for 1-aggregate MDP models.

It is difficult to reveal a clear relation between MAP and ε as the relation between AG and ε. As ε increases, AG increases, however MAP fluctuates. For example, in Table 5.15, consider 0.6 and 0.8 aggregate models, their long-run average costs are same, but they differ from each other in terms of MAP. Interestingly, we observe that the majority of MAP contributions are from the transient states, since even in optimal the best action for these states can be any one of the allowable actions. Therefore, although MAP does not seem to be a good performance indicator, when we only consider the recurrent states in optimal and approximate optimal solutions, it allows anticipate about the performance of the aggregate model.

It is certain that the tractability of the MDP model increases when states are aggregated. The computation time and the memory requirement of corresponding aggregate MDP model are very low compared to optimal solution. In terms of computation times, the decreasing number of aggregate states decreases the number of rows in LP. So, the time required for extraction and solution of LP decreases. For instance, even the 0-aggregated MDP model in Table 5.13 has fewer states than a full MDP $A = 2$. The total computation time for all aggregate models is almost one-fifth of the original models', even a bunch of recursive operations are done to generate aggregated states. Fortunately, for greater ε the required time for generate aggregate states is lower.

Two major facts drive the significant reduction in memory requirement: (i) as the number of states reduces, the size of the probability transition matrix decreases since the number of entries in this matrix is the square of the number of states, (ii) the set of actions of each aggregate state is the intersection of actions of the states that assigned to it, so as the number of actions decreases the number of columns in LP decreases. The notable reduction in memory requirement indicates an approximate optimal policy can be easily found even for very large instances. However, applying the resulting approximate optimal policy to the original MDP again makes difficult to calculate the long-run average cost, as it also calls the full state space. However, even the approximate optimal policy can not evaluate on the original MDP, the approximate optimal long-run average cost certainly provides an upper bound to the optimal cost,

and the approximate optimal policy also gives insight about optimal admission and capacity allocation policy.

Structural results work well with exact approaches such as LP and state space decomposition. However, with ε-aggregation method, action elimination before the model minimization, does not affect the accuracy of the aggregate models in positive way. In Table 5.17 we show the results for the same system worked on in Table 5.13 when the suboptimal actions are first eliminated.

Table 5.17: Computational results on $K = 2$, $A = 4$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$ with action elimination

	$\|\mathcal{S}\|$	$\|\mathcal{D}\|$	Cost	M(MB)	t(sec)	aggt(sec)	MAP	AG
OPT	5625	53290	86.87	132.83	273.65	-	-	-
0-agg	654	1048	100.68	23.79	21.74	0.15	73.44	15.89
0.2-agg	294	341	203.13	3.95	17.48	0.06	74.61	133.82
0.4-agg	286	292	203.73	3.49	16.52	0.05	71.88	134.51
0.6-agg	285	289	204.07	3.46	16.98	0.05	74.93	134.91
0.8-agg	285	289	204.07	3.46	17.37	0.05	76.92	134.91
1-agg	285	289	204.07	3.46	16.60	0.01	76.09	134.91

The elimination of suboptimal actions significantly reduces the chance of assigning each state to existing clusters, since the set of actions of each aggregate state is the intersection of actions of the states that assigned to it, and a state can not be assigned to cluster if the intersection of sets of actions is empty. That is a state s can be considered as candidate for the aggregated state S_m if $\mathcal{D}_s \cap \mathcal{D}_{S_m} \neq \emptyset$. Therefore, the number of states in newly generated aggregate models is greater. Unfortunately, better bounds also are not found for the optimal long-run average cost. However, with the model minimization, the action elimination drags the model to more trivial scenarios, such that almost in every aggregated states there remains a single action, so the computation times are even very low compared to former results. At this point, again, a trade-off appears between the solution quality and the computation time.

We have stated that the second aggregation heuristic that we developed gives faster good quality solutions under scarce and moderate capacities. Therefore, for such cases, it can be a powerful alternative to the ε-aggregation method. Table 5.18 shows

computational results on equally segmented and equal-loaded system with problem parameters $K = 3$, $A = 1$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$. These results compares two aggregation methods in terms of solution quality and computation time.

Table 5.18: Computational results on $K = 3$, $A = 1$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$

	$\|\mathcal{S}\|$	$\|\mathcal{D}\|$	Cost	M(MB)	t(sec)	aggt(sec)	MAP(%)	AG(%)
OPT	1280	7169	0.92	213.17	6.08	-	-	-
0-agg	60	381	1.12	0.96	2.03	0.26	69.38	21.27
0.5-agg	16	86	1.17	0.23	1.19	0.05	72.73	26.02
1-agg	15	102	1.20	0.23	1.20	0.00	70.55	29.58
Tot-job	432	2393	0.93	30.67	4.97	0.02	95.39	0.20

In that case, where $K = 3$ and $A = 1$, $M = 2$ is a moderate server capacity, since at at time at most 2 job requests can be done for the current period. Notice that, since the optimal cost is close to zero, the approximate optimal costs found with ε-aggregate models are not far, but the AG is quite large. The results in Table 5.18 show that the second aggregation method gives fast and near-optimal policies. Fortunately, total job aggregation heuristic is also better than other ε-aggregation in terms of MAP. The disadvantage of this aggregation method is that it generates the same partition of $\mathcal{S}$ for the same K and A values under different costs and arrival probabilities. Similar to structural results, this aggregation method depends only on the problem parameters, such as length of the customer order horizon and the maximum number of arriving job requests for $t + j$.

Unlike the ε-aggregation method, it can be said that the total job aggregation method works also well with the action elimination method. Table 5.19 also shows results of an equally segmented and equally loaded system with $K = 3$ and $A = 1$, $M = 2$. From the results, it is seen that the aggregation method alone is more successful in terms of solution quality. However, considering the solution time and memory requirement, it is seen that the integrated method is more suitable for large instances.

Table 5.19: Computational results on $K = 3$, $A = 1$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$

	$\|\mathcal{S}\|$	$\|\mathcal{D}\|$	Cost	M(MB)	t(sec)	aggt(sec)	MAP(%)	AG(%)
OPT	1280	53290	0.92	213.17	6.08	-	-	-
0-agg	60	381	1.12	0.96	2.03	0.26	69.38	21.27
0.5-agg	16	86	1.17	0.23	1.19	0.05	72.73	26.02
1-agg	15	102	1.20	0.23	1.20	0.01	70.55	29.58
Totjob-agg	432	2393	0.93	30.67	4.56	0.02	78.75	1.08
Totjob-agg & AE	469	1857	1.02	25.19	3.56	0.01	81.09	10.86

In the following two tables, e.g. Table 5.20 and Table 5.21, we compare heuristic and approximate results for the problems that we cannot solve optimally. Table 5.20 shows results of an equally segmented and equally loaded system with $K = 2$ and $A = 6$, $M = 2$ when $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$, while Table 5.21 shows results of an equally segmented and equally loaded system with $K = 3$ and $A = 2$, $M = 2$ when $c_o = 2000$, $c_r = 500$, $c_1^e = 1500$ and $c_2^e = 1000$. Different from other tables, $t(sec)$ from now refers to the time until a policy is found, in seconds.

Table 5.20: Computational results on $K = 2$, $A = 6$, $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$

	$\mathcal{S}$	$\mathcal{D}$	Cost	t(sec)	aggt(sec)
MP	31213	45196	285.26	9.89	-
AP	31213	45196	848.69	10.42	-
1-agg	133	532	285.32	1050.12	0.92
Totjob-agg & AE	6517	19349	240.10	1883.29	0.69

For such larger instances that we can evaluate the heuristic policies on the original MDP, our aggregate MDP models provide better results in terms of long-run average cost. However, it takes long to find approximate optimal policies. Remember the whole procedure, once the aggregate MDP model is generated, the BLP is reformulated with associated aggregate cost and probabilities. The solution of that LP is evaluated on the original MDP respect to 4.14. While the aggregation process alone takes less time than other heuristics, it takes long to find a policy, due to excess com-

Table 5.21: Computational results on $K = 3$, $A = 2$, $M = 2$, $c_o = 2000$, $c_r = 500$, $c_1^e = 1500$ and $c_2^e = 1000$

	$\mathcal{S}$	$\mathcal{D}$	Cost	t(sec)	aggt(sec)
MP	59049	139234	11.59	14.37	-
AP	59049	139234	22.90	16.42	-
1-agg	38	98	9.50	3258.90	3.71
Totjob-agg & AE	11583	30015	8.49	3949.99	3.16

putation requirements of reformulation of LP on aggregate MDP model.

When $K = 4$, the number of states increases exponentially, as shown in Table 5.3. Although we can generate heuristic policies quickly, the dimensionality of the state space incapacitate our computational power to evaluate the long-run average costs of the corresponding heuristic policies. However, for such instances, approximate policies can be generated trough ε-aggregation and total job aggregation methods. In Table 5.22, we give results for an equally segmented and equally loaded system under different cost structures.

Table 5.22: Computational results on $M = 2$, $c_o = 2000$, $c_r = 1500$, $c_1^e = 1000$ and $c_2^e = 500$

		$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{D}\rvert$	Cost	M(MB)	t(sec)	aggt(sec)
$C1$	1-agg	45	68	2430.27	0.14	2801.43	11.72
	Totjob-agg & EA	10149	37425	8.72	3011.42	7552.50	10.02
$C2$	1-agg	46	57	2415.60	0.14	2808.36	11.39
	Totjob-agg & EA	10136	37288	8.82	3002.75	7218.01	9.88
$C3$	1-agg	46	78	2599.02	0.16	2793.31	11.46
	Totjob-agg & EA	10114	37405	16.49	2993.89	7094.48	9.85

For such instances, unsolvable optimally due existing physical memory constraints, we find approximate optimal policies which gives insight about the behavior of the optimal policy and an upper bound for the long-run average cost. Again, our total job aggregation method performs better in terms of solution quality and aggregation computation time, since it can work cordially with action elimination method, and makes use of structural results.

Finally, we can said that for optimal solvable problems we obtain significant improvements in computation times and memory requirements through structural results based action space and state space reduction methods, whereas for intractable systems our problem-specific aggregation method provides quick and high-quality solutions.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

In this thesis, we study the joint problem of capacity allocation and admission control in a queuing system where customers have dynamic priorities and submit their desired service completion time before joining the queue. To the best of our knowledge, extended from the study [5], this is the first study to address the structure and properties of the optimal policy with exact and approximate methods in such queuing systems.

In our system, customers have preferred service completion times, hence customers' satisfaction is directly affected by the service provider's ability to serve the jobs as close as possible to their requested times. A service can be completed either early, or on time using the regular capacity, or by temporarily increasing the capacity through overtime (or equivalently, outsourcing). In the first and last cases, the service provider incurs early service costs and overtime costs, respectively. Besides, when the arriving customer is low-priority, her request can be rejected incurring a rejection cost. The goal is to determine which orders to accept to the queue and how to allocate the capacity, which may include strategic server idling, to serve the jobs waiting in the queue at the minimum cost.

We formulate this problem as a discrete-time infinite-horizon Markov Decision Process (MDP). We propose structural results to characterize a set of suboptimal decisions, and we integrate them into our solution method for the LP formulation for efficient computations. We also develop problem-specific approximation and state space aggregation methods to be able to solve instances of larger size.

Our results show how the structure of the optimal policy varies based on the problem parameters chosen. Under some scenarios, it is relatively easier to find good solutions, and the optimal solutions may even be trivial. These conditions occur when the parameters that directly impact the cost of actions, such as the cost of rejection, early service, overtime, and the server capacity, are extreme. For example, when the server capacity is too scarce to be able to serve requests that are due to the current period, the system controller generally rejects the majority of the arriving low-priority job requests. After fulfilling the remaining requests that are due the current period, the system rarely has any capacity left, and thus the early service option cannot be considered. In contrast, when the server capacity is abundant, all requests can be served on time at the minimum cost. Another similar case is observed when the ratios between rejection and overtime costs, and early service and overtime costs, are very large or small. For instance, when rejection is too costly, the controller decides only to serve on time and early, or when the early service is too costly, the controller only considers overtime and rejection options. In each case, the moderate parameter ratios result in non-trivial solutions that are state-dependent and harder to characterize.

The parameters which determine the number of states have no such obvious effects. As the maximum number of arriving job request for $t + j$ and the length of the customer order horizon increase, the number of rows in the LP, hence the complexity, increases, and finding the optimal admission control and the capacity allocation policy through LP becomes substantially difficult, in some cases, impossible with the available computational resources. In smaller instances, that are solvable with LP, our structural results significantly improve the computation time and the solution method's memory requirement. In larger instances, our structural results and aggregation operators can be used to quickly obtain high-quality solutions compared to simple benchmark heuristics.

The problem introduced in this study can be extended in many different future directions. This study assumes that all customers requests require a unit capacity, result in same overtime costs, and are fulfilled within one time period. Firstly, future work could consider different job types with associated costs, and relax the assumption of service completion time. Another extension would focus on state-dependent arrival probabilities. Although, this new consideration would violate the Markov property;

a somewhat structured optimal policy may exist. Although only two priority classes are examined in this study, more priority classes can be handled later. Also, different transition properties can be addressed by changing the dynamic priority evolution scheme. Customer behaviors such as no-show and last-time cancellation and late fulfillment can be added to increase the model's generality. Lastly, the problem can also be reformulated from the customer's perspective. For instance, when the customer order horizon is short and receiving no service is costly, the customer would assess the trade-off between being in the high-priority class and being rejected.

As we mentioned earlier, our LP formulation's biggest shortcoming is that it does not provide bias-optimal solutions. Hence, LP can be solved under bias optimality condition, and a long-run average cost and bias optimal allocation policy can be stated. In addition, our aggregation operators assume that all states are equally weighted. Instead, one could aim to find better near-optimal solutions by giving higher weight to states recurrent under all deterministic stationary policies. Even though this approach requires more computations, it may increase the solution quality.

REFERENCES

[1] Nielsen, "the Future of Grocery E- Commerce," no. April, pp. 1–35, 2015.

[2] C. Aho, "Curbside pickup is taking over the market." https://usa.inquirer.net/35574/curbside-pickup-is-taking-over-the-market, 2019. Accessed: July 3, 2020.

[3] S. Chopra, "How omni-channel can be the future of retailing," *Decision*, vol. 43, no. 2, pp. 135–144, 2016.

[4] P. N. Danziger, "Walmart leads the soon-to-be $35 billion curbside pickup market." https://www.forbes.com/sites/pamdanziger/2019/04/07/walmart-is-in-the-lead-in-the-soon-to-be-35-billion-curbside-pickup-market/#6a19d996199e, 2019. Accessed: August 10, 2020.

[5] B. Çavdar and T. Işık, "Capacity Allocation in Queuing Systems with Preferred Service Completion Times," pp. 1–25, 2020.

[6] C. H. Papadimitriou and J. N. Tsitsiklis, "The Complexity of Markov Decision Processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

[7] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the Complexity of Solving Markov Decision Problems," pp. 394–402, 2013.

[8] S. Bhulai and G. Koole, "On the Structure of Value Functions for Threshold Policies in Queueing Models," *Journal of Applied Probability*, vol. 40, no. 3, pp. 613–622, 2003.

[9] G. Koole, "Structural results for the control of queueing systems using event-based dynamic programming," *Queueing Systems*, vol. 30, no. 3-4, pp. 323–339, 1998.

[10] G. Koole, *Monotonicity in Markov Reward and Decision Chains: Theory and Applications*. 2006.

[11] L. C. M. Kallenberg, "Classification Problems in MDPs," *Markov Processes and Controlled Markov Chains*, no. i, pp. 151–165, 2002.

[12] E. A. Feinberg and F. Yang, "On polynomial cases of the unichain classification problem for Markov Decision Processes," *Operations Research Letters*, vol. 36, no. 5, pp. 527–530, 2008.

[13] A. P. Trust and A. Probability, "Detecting Optimal and Non-Optimal Actions in Average-Cost Markov Decision Processes Author (s): Jean B . Lasserre Published by : Applied Probability Trust Stable URL : http://www.jstor.com/stable/3215322," vol. 31, no. 4, pp. 979–990, 1994.

[14] L. V. Green, S. Savin, and B. Wang, "Managing patient service in a diagnostic medical facility," *Operations Research*, vol. 54, no. 1, pp. 11–25, 2006.

[15] J. Patrick, M. L. Puterman, and M. Queyranne, "Dynamic multipriority patient scheduling for a diagnostic resource," *Operations Research*, vol. 56, no. 6, pp. 1507–1525, 2008.

[16] N. Ayvaz and W. T. Huh, "Allocation of hospital capacity to multiple types of patients," *Journal of Revenue and Pricing Management*, vol. 9, no. 5, pp. 386–398, 2010.

[17] L. V. Green and S. Savin, "Reducing delays for medical appointments: A queueing approach," *Operations Research*, vol. 56, no. 6, pp. 1526–1538, 2008.

[18] N. Liu, S. Ziya, and V. G. Kulkarni, "Dynamic scheduling of outpatient appointments under patient no-shows and cancellations," *Manufacturing and Service Operations Management*, vol. 12, no. 2, pp. 347–364, 2010.

[19] W. Y. Wang and D. Gupta, "Adaptive appointment systems with patient preferences," *Manufacturing and Service Operations Management*, vol. 13, no. 3, pp. 373–389, 2011.

[20] J. Feldman, N. Liu, H. Topaloglu, and S. Ziya, "Appointment scheduling under patient preference and no-show behavior," *Operations Research*, vol. 62, no. 4, pp. 794–811, 2014.

[21] S. A. Slotnick, "Order acceptance and scheduling: A taxonomy and review," *European Journal of Operational Research*, vol. 212, no. 1, pp. 1–11, 2011.

[22] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of Scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[23] I. Duenyas, "Single Facility Due Date Setting with Multiple Customer Classes," *Management Science*, vol. 41, no. 4, pp. 608–619, 1995.

[24] S. Carr and I. Duenyas, "Optimal admission control and sequencing in a make-to-stock/make-to-order production system," *Operations Research*, vol. 48, no. 5, pp. 709–720, 2000.

[25] L. Mönch, R. Unbehaun, and Y. I. Choung, "Minimizing earliness-tardiness on a single burn-in oven with a common due date and maximum allowable tardiness constraint," *OR Spectrum*, vol. 28, no. 2, pp. 177–198, 2006.

[26] N. Zhao and Z. Lian, "A queueing-inventory system with two classes of customers," *International Journal of Production Economics*, vol. 129, no. 1, pp. 225–231, 2011.

[27] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: John Wiley & Sons, 1994.

[28] O. Hernandez-Lerma and J. B. Lasserre, *Discrete-Time Markov Control Processes*. Springer-Verlag New York, 1996.

[29] J. Bather, "Optimal Decision Procedures for Finite Markov Chains - 3. General Convex Systems.," *Advances in Applied Probability*, vol. 5, no. 3, pp. 541–553, 1973.

[30] K. W. Ross and R. Varadarajan, "Multichain Markov Decision Processes with a Sample Path Constraint: A Decomposition Approach," *Mathematics of Operations Research*, vol. 16, no. 1, pp. 195–207, 1991.

[31] R. Givan, T. Dean, and M. Greig, "Equivalence notions and model minimization in Markov decision processes," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 163–223, 2003.

[32] T. Dean and R. Givan, "Model minimization in Markov decision processes," *Proceedings of the National Conference on Artificial Intelligence*, pp. 106–111, 1997.

[33] R. Ortner, "Pseudometrics for state aggregation in average reward Markov decision processes," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4754 LNAI, pp. 373–387, 2007.

[34] C. Sanderson and R. Curtin, "A User-Friendly Hybrid Sparse Matrix Class in C++," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10931 LNCS, pp. 422–430, 2018.

APPENDIX A

PSEUDO CODES OF AGGREGATION METHODS

Algorithm 1 ε-Aggregation Method

Input: $\varepsilon \geq 0$ and the finite set of states $\mathcal{S}$

Output: The finite set of ε-aggregated states $S_{agg} = S_1, \ldots, S_k$, k is unknown in advance

Set status of each state $s_i \in \mathcal{S}$ as unassigned

Assign the first state s_1 in $\mathcal{S}$ as the seed state of the first aggregate state S_1

Change the status of s_1 from unassigned to assigned

for $i \leftarrow 2$ ***to*** $|\mathcal{S}|$ **do**

  for $j \leftarrow 1$ ***to*** $|S_{agg}|$ **do**

    if $\mathcal{D}_{s_i} \cap \mathcal{D}_{S_j} \neq 0$ **then**

      if $c_c \, \|c(S_j.seed, d) - c(s_i, d')\| < \varepsilon \; \forall d \in \mathcal{D}_{S_j.seed} \; \exists d' \in \mathcal{D}_{s_i}$ **then**

        Assign s_i to the aggregate state S_j

        Change the status of s_i from unassigned to assigned

        $S_j.actions \leftarrow s_i.actions \cap S_j.actions$

        break

      else

        continue

    else

      continue

  if *The status of* s_i *is still unassigned* **then**

    Assign the state s_i in $\mathcal{S}$ as the seed state of the aggregate state S_{j+1}

    Change the status of s_i from unassigned to assigned

for $j \leftarrow 1$ ***to*** $|S_{agg}|$ **do**

  for $i \leftarrow 1$ ***to*** $|S_j|$ **do**

    if $c_p \left\| \sum_{s'' \in \mathcal{S} \setminus \{S_j.seed, S_j.i\}} P(s''|S_j.seed, d) - \sum_{s'' \in \mathcal{S} \setminus \{S_j.seed, S_j.i\}} P(s''|S_j.i, d') \right\| < \varepsilon$ $\forall d \in \mathcal{D}_{S_j.seed} \nexists d' \in \mathcal{D}_{S_j.i}$ **then**

      Change status of $S_j.i$ from assigned to unassigned

      Remove $S_j.i$ from aggregate meta-state $|S_j|$

  $|S_j.actions| \leftarrow S_j.seed.actions$

  for $i \leftarrow 1$ ***to*** $|S_j|$ **do**

    S_j.actions $\leftarrow S_j.i.actions \cap S_j.actions$

for $i \leftarrow 1$ ***to*** $|\mathcal{S}|$ **do**

  if *The status of* s_i *is unassigned* **then**

    for $j \leftarrow 1$ ***to*** $|S_{agg}|$ **do**

      if $\mathcal{D}_{s_i} \cap \mathcal{D}_{S_j} \neq 0$ **then**

        if $c_c \|c(S_j.seed, d) - c(s_i, d')\| < \varepsilon$ *and* $c_p \left\| \sum_{s'' \in \mathcal{S} \setminus \{S_j.seed, S_j.i\}} P(s''|S_j.seed, d) - \sum_{s'' \in \mathcal{S} \setminus \{S_j.seed, S_j.i\}} P(s''|S_j.i, d') \right\| < \varepsilon$ $\forall d \in \mathcal{D}_{S_j.seed} \exists d' \in \mathcal{D}_{s_i}$ **then**

          Assign s_i to the aggregate state S_j

          Change the status of s_i from unassigned to assigned

          $S_j.actions \leftarrow s_i.actions \cap S_j.actions$

          break

        else

          continue

      else

        continue

    if *The status of* s_i *is still unassigned* **then**

      Assign the state s_i in $\mathcal{S}$ as the seed state of the aggregate state S_{j+1}

      Change the status of s_i from unassigned to assigned

Algorithm 2 Total-Job Aggregation Method

Input: The finite set of states $\mathcal{S}$

Output: The finite set of aggregated states $S = S_1, \ldots, S_k$, k is unknown in advance

Set status of each state $s_i \in \mathcal{S}$ as unassigned

Assign the first state s_1 in $\mathcal{S}$ as the seed state of the first aggregate state S_1

Change the status of s_1 from unassigned to assigned

for $i \leftarrow 2$ ***to*** $|\mathcal{S}|$ **do**

- **for** $j \leftarrow 1$ ***to*** $|S|$ **do**
  - **if** $\mathcal{D}_{s_i} \cap \mathcal{D}_{S_j} \neq 0$ **then**
    - **if** $x_i + a_i = x_{S_j.seed} + a_{S_j.seed}$ *that is* s_i *has the same number of jobs in total as* S_j*.seed* **then**
      - Assign s_i to the aggregate state S_j
      - Change the status of s_i from unassigned to assigned
      - $S_j.actions \leftarrow s_i.actions \cap S_j.actions$
      - break
    - **else**
      - continue
  - **else**
    - continue
- **if** *The status of* s_i *is still unassigned* **then**
  - Assign the state s_i in $\mathcal{S}$ as the seed state of the aggregate state S_{j+1}
  - Change the status of s_i from unassigned to assigned