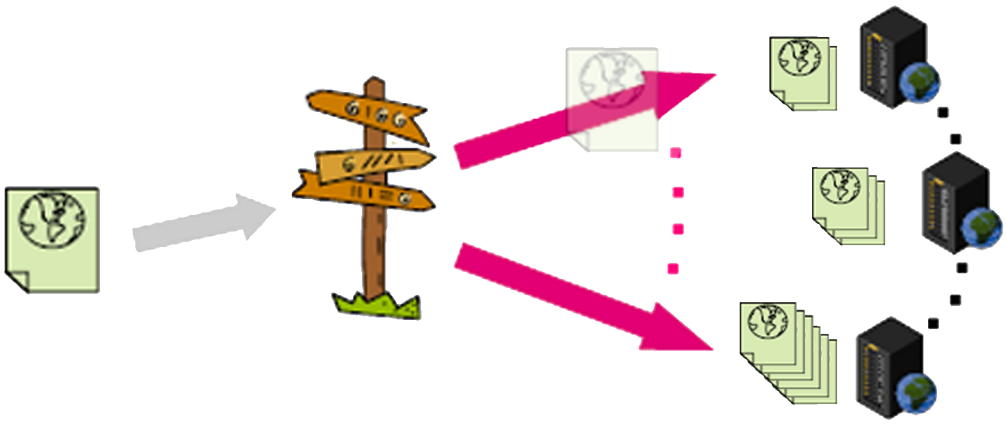
Universiteit Utrecht

Dynamic Load Balancing for High Dimensional Systems

J.S. Janssen
December 2011



Dynamic Load Balancing for High Dimensional Systems

J.S. Janssen (3128474)
Utrecht University, Department of Mathematics
Telekom Innovation Laboratories

24 December 2011

Master's Thesis

Supervisors: Dr. S. Bhulai, VU University Amsterdam
Dr. K. Dajani, Utrecht University
Dr. F. Ciucu, Telekom Innovation Laboratories/TU Berlin
Co-reader: Dr. M.C.J. Bootsma, Utrecht University

Summary

Load balancing is very relevant in networking, where incoming requests (jobs) have to be divided over various servers by dispatchers. The goal is mostly to minimize response times. Research in this field becomes ever more relevant by the current trend where users generate an increasing amount of internet traffic and data centers become larger and larger.
A commonly used load balancing algorithm that uses randomization and performs well, is SQ(d). It sends an arriving job to the shortest of d queues, chosen uniformly at random. In most research this d is kept fixed and its performance is analyzed asymptotically in the number of servers. In this thesis we keep the number of servers finite and apply the theory of Markov decision processes (MDPs) to the load balancing problem. We model the SQ(d) algorithm as a Markov reward process and make it an MDP by introducing communication costs and allowing d to vary. Since MDPs are not scalable, we came up with a heuristic coined Dynamic SQ(d), that outperforms algorithms widely used in practice, such as SQ(2) and Join the Shortest Queue, for a fairly broad range of parameter values.

Keywords: Randomized load balancing, SQ(d), supermarket model, Cloud computing, Markov decision processes, structural properties of the relative value function, partial information models, heuristics.

Preface

In early 2010 I came up with the idea of writing my thesis abroad, preferably at a company. After consulting several professors, it was clear to me that this would not become an easy task. However, these warnings did not discourage me and after a long search and many disappointments, professor Sem Borst from Eindhoven University of Technology gave me the contact details of Florin Ciucu from Telekom Innovation Laboratories[1]/TU Berlin. I would like to thank Sem Borst for being so helpful to someone he never met in person.

Fortunately, Florin was very interested in supervising mathematical students and after some formalities he told me I could come to Berlin to write my master's thesis. Right from the start, I felt that Florin and I would get on well. A feeling that turned out right, partly due to our shared sense of humor and love for playing and watching the beautiful game of football. Throughout my period in Berlin, Florin tried to guide me in my first steps in the world of doing academic research. I will never forget him saying: "*You have to write mathematics, like Barcelona plays football*". Florin, thank you very much for giving me the opportunity to come to Berlin and for guiding me through a fantastic 7 months. I am sure we will meet in the future and finally get to watch a football game together!

I also had to find a supervisor in the Netherlands. I consider myself very lucky to have chosen the course Stochastic Optimization at the VU University Amsterdam in September 2010, because this course was taught by associate professor Sandjai Bhulai. His contagious enthusiasm and open-minded attitude spurred me on to ask him to become my main supervisor in the Netherlands. Although not being a student from his own university, he immediately agreed, for which I am very thankful. I was very happy that after two months in Berlin, you proposed to change tack and approach the problem using Markov decision processes. This gave me a real boost

[1]Until very recently, its name was Deutsche Telekom Laboratories.

since I finally had a concrete goal to work on. I want to thank you very much for your overall supervision and hospitality, but also for all the knowledge you shared with me outside the scope of mathematics.

My third supervisor is senior researcher Karma Dajani from Utrecht University. Thank you for supervising me and helping with the sometimes unorthodox requests from my side, for instance, doing my final presentation more than a month before my actual graduation. I highly appreciate your flexibility in this respect. I also want to thank Martin Bootsma, for taking the time to be second reader from Utrecht University.

I want to thank Yi Lu, assistant professor at the University of Illinois at Urbana-Champaign for taking the time to have a chat with me during her stay in Amsterdam for the IFIP Performance conference last October.

Furthermore, I want to thank my parents and brother for their continuous support and creating the perfect circumstances to flourish. Without you I would not have started nor finished this study. I want to thank my girlfriend Annelie for her patience, understanding and love, and for accepting my absence during the time I was in Berlin.

Finally, I want to thank Tomas Molenaars and Wouter Vink for their great friendship during our study. I very much enjoy being with you guys! Special thanks go to Wouter for designing the front cover of this thesis. My very final thanks goes to KaYin Leung for being such a great study mate during my bachelor.

To close off, I would like to say that I would recommend going abroad to everyone! Most likely you will have a fantastic time and get to know a new culture and lots of new people, but also you learn to appreciate the things you have at home. Moreover, going abroad may result in unexpected events. For me this came in the form of IAESTE, the International Association for the Exchange of Students for Technical Experience, which I came accross in Berlin. This organization is something I could have well used in the Netherlands in my search for an internship abroad and together with Wouter I decided to set up IAESTE in Utrecht.

Have fun reading my thesis!

Jöbke Janssen Utrecht, December 2011

Contents

Chapter 1

Introduction

In this thesis we will apply the theory of Markov decision processes to the problem of randomized load balancing. Most literature on randomized load balancing uses asymptotic techniques to analyze the performance of different algorithms. In our view, the theory of Markov decision processes can add to the existing literature, because of its flexibility and ability to characterize optimal policies in a non-asymptotic setting. To the best of our knowledge this has not been done before and in this thesis we will make a start in this direction.

1.1 Randomized load balancing

The main focus of this thesis is on 'randomized load balancing'. Informally, 'load balancing' is about dividing, as equally as possible, the load of jobs over various servers. 'Randomized' refers to the fact that the algorithm that divides the load, uses, in some way, randomization.
Load balancing is very relevant in networking, where incoming requests (jobs) have to be divided over various servers by dispatchers. The goal is mostly to minimize response times (time from the arrival of a job to its completion) or queue lengths. Research in this field becomes ever more relevant by the current trend where users generate an increasing amount of internet traffic and data centers become larger and larger. In the new 'cloud paradigm', more and more data is stored online in the so-called 'cloud', instead of being stored on our personal devices such as PC, laptop or tablet. One can imagine that this increase in data traffic requires a more comprehensive infrastructure with very large data centers. To illustrate the importance of load balancing, in [21] Amazon and Google predicted that an increase of 500ms in response time for web search would result in a 1.2 % loss of users and revenue.

In general, networks can be very complex but in the mathematical models considered for load balancing they are often represented as in Figure 1.1. This simplified representation allows us to mathematically analyze the system and discover general rules of thumb (supported by simulations) that can be applied when designing networks.

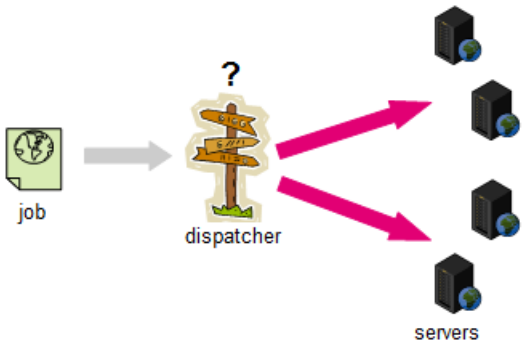


Figure 1.1: This is the simplified network representation that is often used when modelling load balancing problems. The question mark illustrates where load balancing can make a difference.

In this thesis we will mainly concentrate on randomized load balancing as opposed to deterministic load balancing. One reason for this is that, in most current data centres, the number of servers is simply too large to apply a Join the Shortest Queue (JSQ) policy blindly, since the required feedback of this deterministic algorithm might not scale. Secondly, a simple deterministic algorithm such as Round Robin[1] (RR) might result in arbitrarily long server queues when job sizes are not equal (in particular when job sizes are stochastic).
Roughly, randomized load balancing models can be divided into two categories: dynamic (either open or closed) and static. In the *static* model, generally, there are a fixed number of jobs (of the same size) that have to be distributed over the servers and never leave the system. So when all jobs are divided over the servers, the task is done. In case the jobs that complete service get recirculated in the system (the total number of jobs remains fixed), we call this a *closed dynamic* model. This type of model is not very much used in practice, therefore we focus on the *open dynamic* model, where jobs arrive at the system according to some arrival process and leave upon completion.

[1]The Round Robin algorithm sequentially assigns jobs to servers in a fixed order and then starts over again. It does not require any feedback/communication.

1.2 Models and literature

The static and closed dynamic versions of randomized load balancing were first analyzed by Azar et al. [3] using the classical balls-and-bins model. In the balls-and-bins model there are n balls that sequentially have to be divided over M bins, where each ball is placed in the least loaded of d, $d \geq 1$, bins chosen uniformly at random (ties are broken arbitrarily). This 'shortest queue'-like algorithm is called SQ(d) and is intensively studied in randomized load balancing literature. The idea is that d should remain low, to avoid high *communication costs*[2], while performance is measured by the number of balls in the fullest bin (the idea is that a ball represents a job, and the fuller the bin, the longer jobs have to wait).
Azar et al. [3] showed that choosing $d = 2$ resulted in an exponential improvement in performance over choosing $d = 1$. That is, the number of balls in the fullest bin is exponentially higher when $d = 1$ compared to $d = 2$. Increasing d further, improves the performance only slightly; the significant improvement occurs when going from $d = 1$ to $d = 2$. Thus, at the cost of only a small increase in communication[3], there is an exponential improvement in performance. This phenomenon is also called '*the power of two*' and was first observed in [9]. Due to this power of two, SQ(2) is the mostly used SQ(d) algorithm in practice. Independently, Mitzenmacher [14] and Vvedenskaya et al. [24] generalized the result of Azar et al. to the open dynamic version of the randomized load balancing problem. They use the so-called *supermarket model* to analyze this system.

In the supermarket model, a job arrives at a collection of M *subsystems* (each consisting of a server/processor and a queue) according to a Poisson process with rate $\lambda > 0$. The jobs are assumed to be treated according to the first-in-first-out (FIFO) principle and the service times at the M servers are taken to be all exponentially distributed with mean $\mu > 0$. For stability, the *occupation rate* $\rho := \frac{\lambda}{M\mu}$ is assumed to be smaller than 1. See also Figure 1.2.

[2]Instead of 'communication costs' we will also use the term 'sampling costs'. *Sampling* a bin consists of requesting how many balls are present in a bin and receiving the answer. One can imagine that increasing d will lead to an increase in communication costs, since more bins have to be sampled before the ball is allocated to a particular bin. Note that this sampling time is not explicitly taken into account in the models described here, but that implicitly d is imposed to be as low as possible.

[3]In fact, just like RR, SQ(1) requires no communication at all since it assigns a ball to the bin chosen uniformly at random, without having to compare with other bins. For this reason SQ(1) is also called the *Random* algorithm.

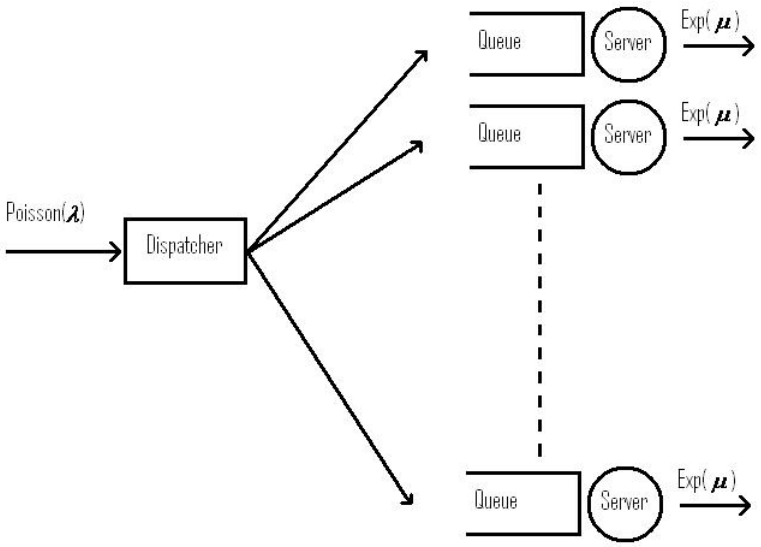


Figure 1.2: The supermarket model. Jobs arrive at a collection of M subsystems, each consisting of a server and a queue, according to a Poisson process with rate $\lambda > 0$. Jobs are served according to the first-in-first-out principle and the service times are exponentially distributed with mean $\mu > 0$. For stability, the occupation rate $\rho := \frac{\lambda}{M\mu}$ is assumed to be smaller than 1.

Many variations of the models and of the algorithms above have been investigated. For instance, in [23] it is shown that breaking ties asymmetrically is better than breaking ties arbitrarily. Mitzenmacher [16] investigates the use of old information and shows how randomness can be of use there. In [17] and [22] it is proved that only a small amount of memory (just like a small amount of choice) can drastically improve the load balancing performance. Ganesh et al. [7] investigate load resampling and migration strategies under a Processor Sharing (PS) service discipline[4]. Clients initially attach to an arbitrary server, but may switch servers independently at random instants of time in an attempt to improve their service rate.
In [14] and [15], instead of exponential service times, constant service times are con-

[4]In the PS service discipline, jobs divide the available processing speed among all jobs that have been assigned to a particular server. The higher the number of jobs that have to share one processor, the lower their actual service rate. This contrasts with the FIFO service discipline where jobs are served in order of arrival.

sidered. In [6], general service time distributions and service disciplines are treated and adopting a particular asymptotic assumption yields very general results. In [20], the authors made an attempt to generalize the above asymptotic results to more general arrival processes and service times. They used a matrix-analytic approach to extend the existing models to Markovian arrival processes and phase-type distributions. Recently, in [13] a completely new class of algorithms was proposed, namely the class of Join the Idle Queue (JIQ) algorithms. The idea is that when servers are empty, they inform the dispatcher about their idleness so that arriving jobs can always be sent to idle servers when available. It turns out that this class of algorithms outperforms the class of SQ(d) algorithms.
All of the above results use asymptotics, letting the number of servers M go to infinity. Certain techniques are used to bound the error between the finite and infinite systems, using extensive simulations to support their claims.

1.3 Research focus

In this thesis we will make a start with introducing Markov decision processes to the research of randomized load balancing. We focus on the SQ(d) algorithms using the supermarket model as framework, but without using asymptotic techniques. Existing literature emphasize the importance of keeping d as low as possible implicitly, but does not make the influence of the corresponding communication costs concrete. This is probably due to the fact that the communication costs are constant for a fixed d. Instead of fixing d, we make it a decision variable, which can vary at every arrival, depending on the costs anticipated. These costs obviously depend on the current state of the system, i.e., the number of jobs in each subsystem, and the occupation rate[5]. Contrary to the traditional algorithms, given an occupation rate we would like to be able to adapt our choice of d to the current state of the system. That is, on the basis of the information we get by sampling subsystems we want to choose a d that best fits the situation, instead of blindly choosing $d = 2$ as is done mostly in practice. To this end, we intend to investigate the trade-off between the communication costs and the costs due to the waiting time in the queues. We think that by taking the communication costs explicitly into account, we can show that varying d per arrival on the basis of the anticipated costs, may lead to a better performance than when restricting ourselves to SQ(2). Observe that this performance is measured not only in terms of the queue sizes, but also in terms of the communication costs. Our ultimate

[5]The occupation rate is a measure for how busy the system is and is formally defined in Figure 1.2.

goal is to come up with an algorithm that outperforms the traditional algorithms, such as RR, Random, SQ(2) and JSQ, over a broad range of parameter values.

1.4 Thesis outline

In Section 2 we will provide the theoretical background needed to understand the models formulated in Section 3. Apart from modelling fixed SQ(d) algorithms as Markov reward processes, we also come up with a Markov decision process (MDP) that models an algorithm that dynamically chooses d per arrival. This MDP produces the optimal d for every state of the system. In Section 4, structural properties of the so-called relative value function of this MDP are investigated. Structural properties, such as non-decreasingness and convexity, may lead to characterization of optimal policies and a decrease in complexity of algorithms that solve the MDP, resulting is lower computation times.
It is important to realize at this point, that for the large data centres we are interested in, it is not feasible to keep track of the exact number of jobs in every subsystem. That is, larger data centres would require more communication to stay up to date. This may lead to congestion of the network, which we do not want because it generally leads to delays. Therefore, throughout this thesis we must assume that we do not have full information on the state of the system. That is, we assume that we do not know exactly how many jobs are present in each subsystem . As a consequence, we cannot directly use the MDP formulated in Section 3, because it uses full information. Therefore, in Section 5 we will develop heuristics that estimate the state of the system on the basis of the information available from earlier samples and use the optimal d from the MDP that corresponds to their estimate. However, since MDPs are generally not scalable[6], we cannot directly use any output from the MDP in our heuristic. Approximating an essential output variable from the MDP, results in a heuristic that we will call "Dynamic SQ(d)". This heuristic is scalable up to at least $M = 30$ systems and outperforms traditional algorithms in certain scenarios. We will close off with a conclusion and ideas for future research.

[6]With *scalable* we mean here that if the dimension of the problem is increased (in our case the dimension relates to the number of servers in the system), the extra computation time does not explode.

Chapter 2

Theoretical background

In this thesis we will make use of the theory of Markov reward chains and Markov decision chains. Central in both these theories is the concept of Markov chains, to which the following section is devoted[1]. After that we will discuss Markov reward chains, followed by Markov decision chains. Subsequently, we will discuss how to transform a Markov process into a Markov chain and we close off with some remarks on computational issues.

2.1 Markov chains

A Markov chain is a discrete random process, where discrete refers in this thesis to both the state space $\mathcal{X}$ (meaning the state space is finite or countable) and the set of times[2] $t \in \{0, 1, 2, \ldots\}$. At fixed times, transitions take place from one state to the next according to transition probabilities

$$p(x, y) \geq 0,$$

with

$$x, y \in \mathcal{X} \text{ and } \sum_{y \in \mathcal{X}} p(x, y) = 1,$$

where $p(x, y)$ should be interpreted as the probability that the chain moves to state y given it is in state x. Let the random variable X_t denote the state at time $t \in \{0, 1, 2, \ldots\}$ with distribution π_t. Then, a Markov chain is a random process

[1] We will explain this theory along the lines of [19].

[2] Some authors also allow for time to take on continuous values. In this thesis we would refer to this as a (semi-)Markov process.

that satisfies the *Markov Property*:

$$P(X_{t+1} = x | X_1 = x_1, X_2 = x_2, \ldots, X_t = x_t) = P(X_{t+1} = x | X_t = x_t). \tag{2.1}$$

That is, given the present state, the past and future states are independent. Consequently, for $t > 0$,

$$\pi_t(x) = \sum_{y \in \mathcal{X}} \pi_{t-1}(y) p(y, x),$$

where π_0 is given.
Note that we have defined the transition probabilities $p(x, y)$ to be independent of t. That is, $p(x, y) := P(X_{t+1} = y | X_t = x)$, for any t and hence the Markov chain we consider in this thesis is called *time-homogeneous* or *stationary*. This time-homogeneity allows the transitions to be described by a single time-independent transition matrix, with entries $P_{xy} := p(x, y)$. So it holds that $\pi_t = \pi_{t-1} P$ from which it follows that $\pi_t = \pi_0 P^t$. This expression is the reason why we call $p^t(x, y) = P^t_{xy}$ the t-step transition probabilities. The vector π_*, for which it holds that

$$\pi_* = \pi_* P, \tag{2.2}$$

is called a *stationary distribution*[3], since, if $\pi_0 = \pi_*$, then $\pi_t = \pi_*$ for all t. Note that $\lim_{t \to \infty} p^t(x, y) \to \pi_*(y)$ for all x. In the remaining part of this thesis, the stationary distribution plays an important role, since we are interested in the long-term average behavior of the systems we consider. In particular, we will assume that the system behaves according to this stationary distribution[4].
Under some conditions one can show this stationary distribution to exist and to be unique. The first condition is that the state space $\mathcal{X}$ is finite:

Condition 1. $|\mathcal{X}| < \infty$

In order to state the other conditions we first need some definitions, where we already assume Condition 1.

Definition 1. *A sequence of states $z_0, z_1, \ldots, z_{k-1}, z_k \in \mathcal{X}$ with the property that $p(z_0, z_1), \ldots, p(z_{k-1}, z_k) > 0$ is called a* **path** *from z_0 to z_k of length k.*

Definition 2. *For a finite state space $\mathcal{X}$, if there is at least one state $x \in \mathcal{X}$, such that there is a path from any state to x, then the chain is called* **unichain** *and state x is called* **recurrent**.

[3] Alternatively, one can say that the system is in a *steady state*, when it behaves according to a stationary distribution.

[4] In the simulations we will do in Chapter 5, we will therefore use a so-called 'warm-up' period, to make sure that the system is in its steady state.

Definition 3. *If there is some recurrent state x for which the greatest common divisor of all paths from x to x is 1, then the chain is called* **aperiodic**.

We now formulate the second and final condition, which, together with Condition 1, forms sufficient conditions for the theorem below. We will assume these conditions throughout this chapter.

Condition 2. *The chain is aperiodic and unichain.*

Now we have the following theorem which we state without proof.

Theorem 2.1.1. *Under the above conditions and for some arbitrary distribution π_0, it holds that*

$$\lim_{t\to\infty} \pi_t = \pi_*,$$

where the distribution π_* (also called the ***limiting distribution****) is the unique solution to system of Equations (2.2) together with the requirement that*

$$\sum_{x\in\mathcal{X}} \pi_*(x) = 1, \tag{2.3}$$

independent of π_0.

Note that writing out the matrix Equation (2.2) gives

$$\pi_*(y) = \sum_{x\in\mathcal{X}} \pi_*(x)p(x,y), \tag{2.4}$$

a system of $|\mathcal{X}|$ equations. The right-hand side of (2.4) is the probability that, starting from stationarity, the chain is in state y the next time instant.
Theorem 2.1.1 allows for an iterative procedure to compute π_*; multiplying some initial distribution repetitively with P, until sequential outcomes do not differ more than some given $\epsilon > 0$, is called *forward recurrence*.

2.2 Markov reward chains

We can extend the Markov chain from the previous section by attaching direct rewards to the states. So for every state $x \in \mathcal{X}$ we have $r(x) \in \mathbb{R}$, the direct award that is obtained each time state x is visited. In this setting, we are not interested in the distribution of X_t, but rather we are interested in $\mathbb{E}r(X_t)$, and particularly in its

limit for $t \to \infty$, as we are interested in the average long term performance of the chain. This limit is given by

$$g = \sum_{x \in \mathcal{X}} \pi_*(x) r(x), \tag{2.5}$$

allowing it to be computed by calculating π_* first and then taking the expectation of r, or by simulating X_t and then computing $\sum_{t=0}^{T-1} r(X_t)/T \to g$ a.s. However, when at a later stage we want to include actions, we will need an alternative method since actions have influence on future behaviour and under simulation/forward recursion only the history is known. Thus, we will need a *backward recursion* method that takes the future into account.
To this end, let $V_T(x)$ be the total expected reward up to $0, \ldots, T-1$ when starting at 0 in x:

$$V_T(x) = \sum_{t=0}^{T-1} \sum_{y \in \mathcal{X}} p^t(x,y) r(y) = \mathbb{E} \sum_{t=0}^{T-1} r(X_t), \tag{2.6}$$

with $X_0 = x$. Note that

$$\begin{aligned} \underbrace{\sum_{x \in \mathcal{X}} \pi_*(x) V_T(x)}_{\text{starting in stationarity}} &= \sum_{x \in \mathcal{X}} \pi_*(x) \sum_{t=0}^{T-1} \sum_{y \in \mathcal{X}} p^t(x,y) r(y) = \sum_{t=0}^{T-1} \sum_{y \in \mathcal{X}} \underbrace{\sum_{x \in \mathcal{X}} \pi_*(x) p^t(x,y)}_{=\pi_*(y) \text{ by (2.2)}} r(y) \\ &= \sum_{t=0}^{T-1} \underbrace{\sum_{y \in \mathcal{X}} \pi_*(y) r(y)}_{=:g \text{ by (2.5)}} = gT. \end{aligned} \tag{2.7}$$

Now define

$$V(x) := \lim_{T \to \infty} [V_T(x) - gT]. \tag{2.8}$$

Then $V(x)$ is the total expected difference in reward between starting in x (definition $V_T(x)$) and starting in stationarity (see (2.7)). Since $\lim_{t \to \infty} p^t(x,y) \to \pi_*(y)$ for all x and Equation (2.5) holds, we see from (2.6) that $\lim_{T \to \infty} V_T(x)/T = g$.
We can calculate $V_{T+1}(x)$ in two different ways. On the one hand,

$$V_{T+1}(x) = V_T(x) + \sum_{y \in \mathcal{X}} \pi_T(y) r(y),$$

for π_0 with $\pi_0(x) = 1$. Using (2.5) and Theorem 2.1.1 (i.e. $\pi_T \to \pi_*$), we have

$$V_{T+1}(x) = V_T(x) + g + o(1), \tag{2.9}$$

where $o(1) \to 0$ for $T \to \infty$[5]. On the other hand, we have the following recursive formula:

$$V_{T+1}(x) = r(x) + \sum_{y \in \mathcal{X}} p(x, y) V_T(y). \tag{2.10}$$

Combining expressions (2.9) and (2.10) yields

$$V_T(x) + g + o(1) = r(x) + \sum_{y \in \mathcal{X}} p(x, y) V_T(y).$$

Subtracting gT from both sides and taking $T \to \infty$ gives the so-called *Poisson equations*:

$$V(x) + g = r(x) + \sum_{y \in \mathcal{X}} p(x, y) V(y). \tag{2.11}$$

Equation (2.11) does not have a unique solution, since if V is a solution, then also $V+C$ is. To get a unique solution, we can either add the condition $\sum_{y \in \mathcal{X}} \pi_*(y) V(y) = 0$ (this conserves the interpretation of $V(x)$ being the expected difference in reward between starting in state x and starting in stationarity), or we could take $V(x_0) = 0$ for some reference state x_0 (here the above mentioned interpretation of V does not hold)[6]. Under the conditions from Section 2.1, Proposition 8.2.1 in [19] states that the resulting g is independent of the initial state.

2.2.1 Value iteration

In general, Equation (2.11) is hard to solve. To this end, often a recursion algorithm that exploits recursion (2.10), is used to find V and g. This algorithm is called *value iteration* and the idea is as follows. Since

$$V_{T+1}(x) - V_T(x) = V_T(x) + g + o(1) - V_T(x) = g + o(1) \to g,$$

and

$$V_T(x) - V_T(y) = [V_T(x) - gT] - [V_T(y) - gT] \to V(x) - V(y), \tag{2.12}$$

[5] $f(T) \in o(1)$ if $\lim_{T \to \infty} \frac{f(T)}{1} = 0$.

[6] For our computations later in this thesis, we choose the latter option, since this is the simpler of the two and since we are mainly interested in g.

for $T \to \infty$, we can obtain all values we are interested in, by computing V_T for T large enough[7]. We compute V_T using (2.10), where for initialization one usually takes $V_0 \equiv 0$. The iteration ends when, for some prefixed $\epsilon > 0$ and for all $x \in \mathcal{X}$, it holds that there exists a g such that

$$g - \frac{\epsilon}{2} \leq V_{T+1}(x) - V_T(x) \leq g + \frac{\epsilon}{2},$$

for some T. This is equivalent to

$$\text{span}(V_{T+1}(x) - V_T(x)) \leq \epsilon.$$

See Appendix A.4 for a concrete example of the value iteration algorithm.

2.3 Markov decision chains

Finally, we add decisions to the Markov reward chain, turning it into a Markov decision chain. The decisions (also called actions) come from a decision space $\mathcal{D}$ and the idea is that depending on the state $X_t \in \mathcal{X}$, a decision $D_t \in \mathcal{D}$ is selected according to some policy $R : \mathcal{X} \to \mathcal{D}$, that is, $D_t = R(X_t)$. Consequently, both the transition probabilities and the rewards depend on the decisions. So we have $p(x, d, y)$ and $r(x, d)$, the probability of going from x to y and the reward in state x, respectively, when decision d is taken. In this thesis we will make the assumption that the decision space is finite:

Condition 3. $|\mathcal{D}| < \infty$.

Moreover, we have to adapt condition 2 as follows:

Condition 4. *For every fixed policy R, the chain is unichain and aperiodic (the recurrent state may depend on R).*

Define $V_T^R(x)$ as the total expected reward in $0, \ldots, T-1$, under policy R when starting at time 0 in x. We are now interested in finding the policy that maximizes the average expected long-run reward, i.e., we are interested in finding $\arg\max_R \lim_{T\to\infty} V_T^R(x)/T$. Note that this maximum is well defined because the number of different policies is at

[7] To compute $V(x)$, we take $y = x_0$ in (2.12), with $V(x_0) = 0$.

most $|\mathcal{X}||\mathcal{D}| < \infty$. By the very definition of optimality, it holds that for the optimal policy R^*

$$r(x, R^*(x)) + \sum_{y \in \mathcal{X}} p(x, R^*(x), y) V^{R^*}(y) = \max_{d \in \mathcal{D}} \{ r(x, d) + \sum_{y \in \mathcal{X}} p(x, d, y) V^{R^*}(y) \}$$

Also, by the Poisson equations (2.11):

$$V^{R^*}(x) + g^{R^*} = r(x, R^*(x)) + \sum_{y \in \mathcal{X}} p(x, R^*(x), y) V^{R^*}(y).$$

Combining the above two equations we find the so-called *optimality equations* or *Bellman equations*:

$$V^{R^*}(x) + g^{R^*} = \max_{d \in \mathcal{D}} \left\{ r(x, d) + \sum_{y \in \mathcal{X}} p(x, d, y) V^{R^*}(y) \right\}. \tag{2.13}$$

Note that

$$R^*(x) \in \operatorname*{argmax}_{d \in \mathcal{D}} \left\{ r(x, d) + \sum_{y \in \mathcal{X}} p(x, d, y) V(y) \} \right\}.$$

We can find the optimal policy R^* by extending the value iteration algorithm to Markov decision chains, by including decisions in the recursion of equation (2.10):

$$V_{T+1}(x) = \max_{d \in \mathcal{D}} \left\{ r(x, d) + \sum_{y \in \mathcal{X}} p(x, d, y) V_T(y) \right\}. \tag{2.14}$$

The same stop criterion as in the Markov reward case is used and the optimal policy R^* is recovered by choosing for each $x \in \mathcal{X}$:

$$R^*(x) \in \operatorname*{argmax}_{d \in \mathcal{D}} \left\{ r(x, d) + \sum_{y \in \mathcal{X}} p(x, d, y) V_T(y) \} \right\},$$

with T the iteration number for which the stop criterion was fulfilled.
In Sections 8.5.2-8.5.4 of [19], it is proven that under conditions 1, 3 and 4, the value iteration algorithm converges to an ϵ-optimal solution.

2.4 Continuous time: (semi-)Markov processes

In this section we will generalize the above theory to the case where the time it takes to move from a state x to the next is a random variable $T(x)$, instead of being equal to 1. We define $\tau(x) := \mathbb{E}T(x)$, the expectation of $T(x)$, where we assume that $0 < \tau < \infty$. For general random variables $T(x)$, this is called a *semi-Markov process*. When $T(x)$ is exponentially distributed it is called a *Markov process*. We will see that Markov processes are convenient to work with because we can transform them so that the previous theory on discrete Markov chains becomes relevant again.

2.4.1 semi-Markov processes

If we look at the semi-Markov process only at the jump times (the moments it changes state), then we observe the so-called *embedded Markov chain*. Let this Markov chain have stationary distribution π_*. Additionally, consider the stationary distribution over time, i.e., the time-limiting distribution that the chain is in a certain state, call it ν_*. Then ν_* is specified by:

$$\frac{\nu_*(x)}{\nu_*(y)} = \frac{\tau(x)\pi_*(x)}{\tau(y)\pi_*(y)},$$

from which one can easily find

$$\nu_*(x) = \frac{\tau(x)\pi_*(x)}{\sum_{y\in\mathcal{X}} \tau(y)\pi_*(y)}. \tag{2.15}$$

2.4.2 Markov processes

In the special case that all transition times $T(x)$ are exponentially distributed, we have a Markov process. However, a Markov process is often defined through its transition rates $\lambda(x, y)$. Thus, the time until the process moves from x to y is exponentially distributed with rate $\lambda(x, y)$, unless a transition to another state occurs first. Since the minimum of independent exponentially distributed random variables is again exponentially distributed, with as rate the sum of the constituent rates, we find that $T(x)$ is exponentially distributed with rate $\sum_{y\in\mathcal{X}} \lambda(x, y)$. Moreover, $p(x, y) = \lambda(x, y)/\sum_{z\in\mathcal{X}} \lambda(x, z)$. Defined this way, Markov processes are indeed special cases of semi-Markov processes.

2.4.3 Uniformization

As mentioned above, we can transform Markov processes such that we can use the theory of discrete Markov chains again. This concept is called *uniformization* (see Section 11.5 of [19]). The idea is as follows. In general not all $T(x)$ need to be equally distributed for all x, but adding 'dummy' transitions to make the rates out of states constant, makes this possible. Indeed, let γ be such that $\sum_{y\in\mathcal{X}} \lambda(x,y) \leq \gamma$ for all $x \in \mathcal{X}$[8]. Define a new process with rates $\lambda'(x,y)$ as follows. Firstly, take $\lambda'(x,y) = \lambda(x,y)$ for all $x \neq y$. In each state x with $\sum_{y\in\mathcal{X}} \lambda(x,y) < \gamma$, add a dummy transition from x to x such that the rates sum up to γ: $\lambda'(x,x) = \gamma - \sum_{y\neq x} \lambda(x,y)$ for all $x \in \mathcal{X}$. This newly constructed process has expected transition times τ' such that $\tau'(x) = 1/\gamma$ for all $x \in \mathcal{X}$. Now it follows from Equation (2.15) that $\pi'_* = \nu'_*$ because $\tau'(x) = \tau'(y)$ for all $x, y \in \mathcal{X}$, and we can use the theory on Markov chains above. The concept of uniformization easily generalizes to Markov reward and Markov decision process.

2.5 Computational issues

Most sytems in real life seem finite, but many models in mathematical applications have a countable (and thus infinite) state space. Reasons for this are for instance that infinite systems behave nicer than finite systems or that bounds are not exactly known. For instance, in queueing systems we often assume an infinite buffer length because the exact buffer length is assumed to be large but not known exactly. Although in reality infinite buffers do not exist, the countable state space often allows for nice mathematical formulas. Also, the Markov reward and decision models we initially formulate in Chapter 3 assume a countable state space.
However, although we have good reasons to prefer infinite state spaces in certain cases, we need a finite state space as soon as we want to compute performance measures or optimal policies. Concretely, we have to change a countable state space $\mathcal{X}$, into a finite state space $\bar{\mathcal{X}}$ (i.e., $|\bar{\mathcal{X}}| < \infty$) to make computations possible. Although this adaptation may seem as a restriction, it fortunately brings us in the situation of the theory developed earlier in this chapter. Moreover, in this way we basically model the same problem, but then with finite buffers. The way we approach this in this thesis is as follows.

Given a model with countable state space $\mathcal{X}$ and transition probabilities p, we ap-

[8]Since we assume that $\mathcal{X}$ is finite, as a consequence we know that the transition rates are bounded.

proximate it by a series of models with finite state spaces $\mathcal{X}^K$, such that $\mathcal{X}^K \subset \mathcal{X}^{K+1}$ and $\lim_{K\to\infty} \mathcal{X}^K = \mathcal{X}$. Let the K^{th} model have transition probabilities p^K, given as follows. For each $x \in \mathcal{X}^K$

$$\begin{aligned} p^K(x,y) &= p(x,y) \text{ for } x \neq y \in \mathcal{X}^K, \\ p^K(x,y) &= 0 \text{ for } y \notin \mathcal{X}^K, \\ p^K(x,x) &= p(x,x) + \sum_{y \notin \mathcal{X}^K} p(x,y). \end{aligned}$$

Note that in the model with transition probabilities p^K, there are no transitions from $\mathcal{X}^K$ to $\mathcal{X} \setminus \mathcal{X}^K$. We call K the *truncation level* of the model.
The performance measure of the approximating models should, for large enough K, converge to the performance measure of the original model (i.e., for large enough K, boundary effects should become negligible). There is however, relatively little theory about these types of results. In practice one compares g^K with g^{K+1} for increasing K, until this difference is small enough.

Remark 1. *Curse of dimensionality*

Apart from the computational issues discussed above, Markov decision processes suffer from the so-called *curse of dimensionality*. To explain this phenomenon, assume a model with state space $\mathcal{X} = \{(x_1, \ldots, x_M) | x_i = 0, 1, \ldots, K\}$. The total number of different states is then $|\mathcal{X}| = (K+1)^M$, which grows exponentially in the dimension M. Since most practical problems have multi-dimensional state spaces, this phenomenon really poses a problem. Indeed, for the practical execution of the value iteration algorithm we need in memory at least one double for every state, resulting in, for $M = K = 10$, ± 100 GB of memory[9], without even thinking about the processing requirements that would be necessary.

As we are interested in the performance of high dimensional systems in this thesis, one of the biggest challenges will be to find a way of dealing with this scalability problem.

[9] See [5].

Chapter 3

Models

In this chapter we use the Markov decision theory developed in Chapter 2 to model different algorithms in the supermarket model setting. Firstly, we will model the SQ(2) policy as a Markov reward process and extend this to the SQ(d) policy. Secondly, we introduce decisions and formulate a Markov decision process that models an SQ(d) algorithm that chooses d dynamically. This results in an optimal policy, i.e., for every state x, the MDP chooses the optimal d. Since for a fixed d the communication costs are constant, we will only include these costs when formulating the Markov decision process, since d is not fixed there. Finally, we show that the MDP indeed outperforms fixed d policies and illustrate the 'power of two', to which we refered in the introduction.
For completeness, in Appendix A.2 we formulate a Markov reward process for the Round Robin algorithm.

3.1 A Markov reward model for the SQ(2) policy

We start with the SQ(2) policy. Recall that, for an arriving job, this policy uniformly samples $d = 2$ queues and assigns the job to the subsystem with the least number of customers. We introduce a multidimensional Markov reward process to model this system. In this model, we define the state variables a bit differently than might be expected. This choice of definition will, however, be of convenience when defining the transition probabilities below.

Let x_i be the number of customers in the ith shortest queue, $x = (x_1, \ldots, x_M) \in \mathbb{N}_0^M$. Note that this implies that

$$x_1 \leq x_2 \leq \cdots \leq x_M. \tag{3.1}$$

Hence we define the state space $\mathcal{X}$ as follows

$$\mathcal{X} = \{s \in \mathbb{N}_0^M | s_1 \leq s_2 \leq \cdots \leq s_M\}.$$

Our goal is to minimize the maximum queue length[1], so we should choose the cost function as

$$c(x) := x_M.$$

We now want to calculate the transition probabilities of going from one state x to a new x'. We will uniformize the system and scale the parameters without loss of generality such that $\lambda + M\mu = 1$. This allows us to interpret the rates as probabilities.

First we discuss the transition probabilities for when a job arrives. Here the somewhat odd choice of state variables comes in handy. Note that due to the scaling, we have that with probability λ a job arrives. Recall that at an arrival, we sample $d = 2$ subsystems and send the job to the shortest of the two. If we want to send the job to the i^{th} smallest subsystem, then this subsystem should be among the $d = 2$ sampled subsystems. The other sampled subsystem must be any subsystem with a larger number of jobs. Due to the ordering of x as in (3.1), this last subsystem is any $j > i$. Since, with probability $\frac{1}{M}$, we pick subsystem i first and then, with probability $\frac{M-i}{M-1}$, any $j > i$, we have that the probability that an arriving job is sent to subsystem i is

$$\frac{2(M-i)}{M(M-1)}, \tag{3.2}$$

where the 2 in front comes from the fact that we can also first pick any $j > i$ (with probability $\frac{M-i}{M}$) and then pick i (with probability $\frac{1}{M-1}$). So the probability that a job arrives and is sent to subsystem i is

$$2\lambda \frac{M-i}{M(M-1)}. \tag{3.3}$$

Next, we consider the departure probabilities, which are much simpler. Let $\mathbb{I}\{x_i > 0\}$ denote the indicator function of the event that $x_i > 0$. Since the probability of a departure is $\mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\}$ (jobs can only leave a non-empty subsystem), the

[1]Throughout this thesis, with 'queue length' we will always mean the total number of jobs in a subsystem. That is, the number of jobs in the queue waiting to be served, plus the job in service. Minimizing the maximum queue length makes sure that the dispatcher balances the load equally over all subsystems, since otherwise the maximum queue length would increase. Alternative objective functions, such as minimizing the *average* queue length, are of course also possible.

probability of a departure from subsystem i is $\mu\mathbb{I}\{x_i > 0\}$.
Finally, the transition probabilities due to the uniformization are

$$1 - \left(\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\left\{x_i > 0\right\}\right),$$

which basically represents a transition from x to x (so no arrival or departure). Note that this probability is only positive in states with idle servers.
Before giving the corresponding Poisson equations, we should observe the following. Let e_i be the i^{th} unit vector and note that it might occur that $x+e_i$ (representing an arrival to subsystem i) or $x-e_i$ (representing a departure from subsystem i) results in a new state variable that does not obey (3.1). For example, say $M = 4$ and $x = (1, 2, 2, 5)$. Then $x' = x + e_2 = (1, 3, 2, 5)$, but $x'_2 > x'_3$, which contradicts (3.1). Similarly, $x'' = x - e_3 = (1, 2, 1, 5)$, which again contradicts (3.1), since $x''_2 > x''_3$. A possible way of dealing with this, is to apply a sorting function σ, which sorts any $x \in \mathbb{N}_0^M$ such that it satisfies (3.1). This results in the following Poisson equations

$$\begin{aligned} V(x) + g = x_M &+ \lambda \sum_{i=1}^{M-1} \frac{2}{M} \frac{M-i}{M-1} V(\sigma(x + e_i)) \\ &+ \mu \sum_{i=1}^{M} \mathbb{I}\left\{x_i > 0\right\} V(\sigma(x - e_i)) \\ &+ \left[1 - \left(\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\left\{x_i > 0\right\}\right)\right] V(x), \end{aligned} \tag{3.4}$$

where $\sigma(x')$ denotes that, possibly, necessary sorting has been applied to x' in order to ensure that (3.1) is satisfied. Note that it might happen that $\sigma(x+e_j) = \sigma(x+e_k)$ for $j, k = 1, \ldots, M$ and $j \neq k$ or $\sigma(x-e_j) = \sigma(x-e_k)$ for $j, k = 1, \ldots, M$ and $j \neq k$. This does, however, not change anything in the above Poisson equations. That is, the probability of going to, say, state $x' = \sigma(x + e_i)$, is just the sum over all $\frac{2\lambda}{M}\frac{M-j}{M-1}$, $j = 1, \ldots, M$ such that $\sigma(x + e_j) = x'$ (and similarly if $x' = \sigma(x - e_i)$).

Thus, we will perform value iteration with the following recursion

$$\begin{aligned} V_{n+1}(x) = x_M &+ \lambda \sum_{i=1}^{M-1} \frac{2}{M} \frac{M-i}{M-1} V_n(\sigma(x+e_i)) \\ &+ \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} V_n(\sigma(x-e_i)) \\ &+ \left[1 - \left(\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\}\right)\right] V_n(x). \end{aligned} \tag{3.5}$$

Initially, one usually takes $V_0 = 0$.

3.2 Generalizing: A Markov reward model for the SQ(d) policy

Generalizing SQ(2) to SQ(d) comes down to changing the transition probabilities for an arriving job. Following a similar reasoning as in the previous section for equation (3.2), the probability that given an arriving job, the job is sent to the i^{th} smallest queue is

$$k(M,i,d) := \frac{d(M-i)(M-i-1)\cdots(M-i-d+2)}{M(M-1)\cdots(M-d+1)}, \tag{3.6}$$

for $i \leq M-d+1, d \in \{2,\ldots,M\}$. Note that for $d=2$ this indeed simplifies to (3.2). For $d=1, i \leq M$ this probability should be $\frac{1}{M}$. Moreover, $k(M,i,d)$ is not defined for $i > M-d+1$ but should equal 0. Therefore, it is better to rewrite $k(M,i,d)$ as

$$k(M,i,d) := \begin{cases} d\frac{(M-i)!(M-d)!}{(M-i-d+1)!M!} & \text{if } i \leq M-d+1, \\ 0 & \text{otherwise,} \end{cases} \tag{3.7}$$

for all $d \in \{1, \ldots, M\}$. Lemma 3.2.1 below shows that these are indeed probabilities. The corresponding Poisson equations (compare with (3.4)), then generalizes to

$$\begin{aligned} V(x) + g = x_M &+ \lambda \sum_{i=1}^{M-d+1} k(M,i,d) V(\sigma(x+e_i)) \\ &+ \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} V(\sigma(x-e_i)) \\ &+ \left[1 - \left(\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\}\right)\right] V(x). \end{aligned} \tag{3.8}$$

Hence, we will perform value iteration with the following recursion

$$\begin{aligned} V_{n+1}(x) = x_M &+ \lambda \sum_{i=1}^{M-d+1} k(M,i,d) V_n(\sigma(x+e_i)) \\ &+ \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} V_n(\sigma(x-e_i)) \\ &+ \left[1 - \left(\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\}\right)\right] V_n(x). \end{aligned} \tag{3.9}$$

We have to make an important comment at this point.

Remark 2.

When using the transition probabilities $k(M,i,d)$ as defined in (3.7), it seems as if two subsystems with an equal number of jobs do not have equal probability of receiving an arriving job. That is, due to the fact that we sort our state variables (even if two subsystems have an equal number of jobs), one is placed after the other and hence gets a different probability assigned by $k(M,i,d)$. For instance, let $M = 4$, $d = 2$ and $x = (1,4,4,6)$. Then $x_2 = 4 = x_3$, and both subsystems should have equal probability of receiving an arriving job. However, $k(4,2,2) = \frac{1}{3} \neq \frac{1}{6} = k(4,3,2)$. The reason why this does not pose any problems in the models above, is again the fact that we apply sorting. Indeed, note that in both cases, the resulting $x' = \sigma(x+e_i)$ is the same for $i = 2,3$. That is, $x' = \sigma(x+e_2) = \sigma(x+e_3) = (1,4,5,6)$.
In unsorted systems (as we will consider in Chapter 5), the $k(M,i,d)$ can be used to compute the real probability for a subsystem to receive an arriving job. To this end, let y for the moment be the state variable in an unsorted system, e.g., $y = (4,6,1,4)$.

Define $p_i^M(y,d)$ as the probability that an arriving job is sent to subsystem i in state $y = (y_1, \ldots, y_M)$ when SQ(d) is applied. Let $S_i(y)$ be the set of subsystems with the same number of jobs as subsystem i, i.e., $S_i(y) = \{j | y_i = y_j\}$. For example, $S_1(y) = \{1,4\}$ and $S_2(y) = \{2\}$. Then, the observation

$$\sum_{j \in S_i(y)} k(M,j,d) = \sum_{j \in S_i(y)} p_j^M(y,d), \tag{3.10}$$

will lead to a concrete expression of $p_i^M(y,d)$ below. To illustrate (3.10), consider again $x = (1,4,4,6)$ and assume $d = 2$. Note that $S_1(x) = \{1\}, S_2(x) = S_3(x) = \{2,3\}$ and $S_4(x) = \{4\}$. Hence, it should hold that subsystems 2 and 3 have the same probability of receiving an arriving job, i.e., $p_2^4(x,2) = p_3^4(x,2)$. Observe that $p_1^4(x,2) = \frac{1}{2}, p_4^4(x,2) = 0$ and indeed $p_2^4(x,2) = p_3^4(x,2) = \frac{1}{4}$ should hold. Finally, $k(4,1,2) = \frac{1}{2}, k(4,4,2) = 0$ and recall $k(4,2,2) = \frac{1}{3} \neq \frac{1}{6} = k(4,3,2)$. Then it can easily be seen that (3.10) holds for $i = 1,4$ and also for $i = 2,3$, since

$$p_2^4(x,2) + p_3^4(x,2) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} = \frac{1}{3} + \frac{1}{6} = k(4,2,2) + k(4,3,2).$$

This leads to the following definition.

Definition 4. *Fix $M > 0$, $d \in \{1, \ldots, M\}$ and $i \in 1, \ldots, M$. Given a (not necessarily sorted) state y, the probability that an arriving job is sent to subsystem i in state $y = (y_1, \ldots, y_M)$ when SQ(d) is applied, is*

$$p_i^M(y,d) := \frac{1}{|S_i(y)|} \sum_{j \in S_i(y)} k(M,j,d), \tag{3.11}$$

where $S_i(y) = \{j | y_i = y_j\}$ and $k(M,j,d)$ is defined in (3.7). For ease of notation we will often write $p_i(y,d)$ instead of $p_i^M(y,d)$.

The following Lemma states that the $k(M,i,d)$ defined in (3.7) are probabilities, from which it directly follows that the $p_i(y,d)$ are probabilities.

Lemma 3.2.1. *Given $M > 0$. Then the $k(M,i,d)$ are probabilities, i.e., $0 \leq k(M,i,d) \leq 1$ for $d \in \{1, \ldots, M\}$ and $i \leq M - d + 1$, and*

$$\sum_{i=1}^{M-d+1} k(M,i,d) = 1, \tag{3.12}$$

for all $d \in \{1, \ldots, M\}$.

Proof. First of all, note that all terms involved are non-negative, since $d \leq M$ and $i \leq M - d + 1$ (recall $0! = 1$). Hence, $k(M, i, d) \geq 0$. Next, observe that $k(M, i, d)$ can also be written as

$$k(M, i, d) = \frac{\binom{M-i}{d-1}}{\binom{M}{d}}.$$

To show that $k(M, i, d) \leq 1$, it thus suffices to show that $\binom{M-i}{d-1} \leq \binom{M}{d}$ holds[2]. To this end, observe that $\binom{M-i}{d-1} \leq \binom{M-1}{d-1}$, since $i \geq 1$ and $f_k(n) := \binom{n}{k}$ is increasing in n for fixed $k \leq n$. The equality $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ then yields

$$\binom{M-i}{d-1} \leq \binom{M-1}{d-1} \leq \frac{d}{M}\binom{M}{d} \leq \binom{M}{d},$$

since $d \leq M$.
Finally, we have to show that (3.12) holds. Well,

$$\begin{aligned}
\Sigma &:= \sum_{i=1}^{M-d+1} k(M, i, d) \\
&= \sum_{i=1}^{M-d+1} d\frac{(M-i)!(M-d)!}{(M-i-d+1)!M!} \\
&= d\frac{(M-d)!}{M!} \sum_{i=1}^{M-d+1} \frac{(M-i)!}{(M-i-d+1)!}\frac{(d-1)!}{(d-1)!} \\
&= d(d-1)!\frac{(M-d)!}{M!} \sum_{i=1}^{M-d+1} \frac{(M-i)!}{(M-i-d+1)!(d-1)!} \\
&= \frac{d!(M-d)!}{M!} \sum_{i=1}^{M-d+1} \binom{M-i}{d-1} \\
&= \binom{M}{d}^{-1} \sum_{i=1}^{M-d+1} \binom{M-i}{d-1}.
\end{aligned}$$

So to prove that Σ is equal to 1, it should hold that

$$\binom{M}{d} = \sum_{i=1}^{M-d+1} \binom{M-i}{d-1}. \tag{3.13}$$

[2]In fact, we do not need to show that $k(M, i, d) \leq 1$ directly, since it follows immediately from $k(M, i, d) \geq 0$ together with $\sum_{i=1}^{M-d+1} k(M, i, d) = 1$.

To this end, we will use induction over $M \geq d$ and the standard recursion (see [1], p. 822)

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \text{ for all } n, k > 0, \tag{3.14}$$

with initial values $\binom{n}{0} = 1$ for all integers $n \geq 0$, and $\binom{0}{k} = 0$ for all integers $k > 0$. For $M = d$, we have that $\binom{M}{d} = \binom{d}{d} = 1 = \binom{d-1}{d-1} = \sum_{i=1}^{M-d+1} \binom{M-i}{d-1}$. Now suppose that for $M > d$ (3.13) holds. Then, for $M + 1$:

$$\begin{aligned}
\binom{M+1}{d} &\overset{(3.14)}{=} \binom{M}{d-1} + \binom{M}{d} \\
&= \binom{M}{d-1} + \underbrace{\sum_{i=1}^{M-d+1} \binom{M-i}{d-1}}_{\text{by the induction hypothesis}} \\
&= \sum_{i=1}^{M+1-d+1} \binom{M+1-i}{d-1},
\end{aligned}$$

which proves (3.13) and thus (3.12).

□

3.3 Markov decision model for a dynamic SQ(d) policy

In the previous sections we have not taken the costs of communication into account since these costs are fixed for a fixed d. If we, however, let go of a fixed SQ(d) policy and let the algorithm choose the number of queues sampled at every arrival dynamically, we have a dynamic SQ(d) policy. In order for this to make sense, we need to take into account the cost of communication. Firstly, we still have the regular costs due to the maximum queue length x_M. Let these costs be captured by the function $f_1(x_M)$. Let the cost of communication be given by a function of d, say $f_2(d)$ (f_1 and f_2 should be some sort of utility functions, having a common unit of cost). Note that increasing d will lead to extra costs immediately through $f_2(d)$, but will generally have a diminishing effect on $f_1(x_M)$, since the opportunity cost of sampling fewer queues is that x_M and hence $f_1(x_M)$ have a higher probability of increasing. Thus there is obviously a trade-off between the two, something which a Markov decision process is able to take into account.

The resulting optimality equation will be very similar to the Poisson equations of the fixed SQ(d) policy. Basically, we extend Equation (3.8) by minimizing over d, which comes down to:

$$\begin{aligned} V(x) + g = \min_{d \in \mathcal{D}} \left\{ f_2(d) + \lambda \sum_{i=1}^{M-d+1} k(M,i,d) V(\sigma(x+e_i)) \right\} \\ + f_1(x_M) + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} V(\sigma(x-e_i)) \\ + \left[1 - \left(\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} \right) \right] V(x). \end{aligned} \tag{3.15}$$

Consequently, we can perform value iteration with the recursion

$$\begin{aligned} V_{n+1}(x) = \min_{d \in \mathcal{D}} \left\{ f_2(d) + \lambda \sum_{i=1}^{M-d+1} k(M,i,d) V_n(\sigma(x+e_i)) \right\} \\ + f_1(x_M) + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} V_n(\sigma(x-e_i)) \\ + \left[1 - \left(\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} \right) \right] V_n(x). \end{aligned} \tag{3.16}$$

Remark 3.

As mentioned in Chapter 2, in practice we will truncate the state space $\mathcal{X}$. Hence the state space is finite, as is the action space $\mathcal{D} = \{1, \ldots, M\}$. Due to the uniformization we applied, there is a path from x to x of period 1 and our model is aperiodic. Since for a stable system and every policy R, the empty state $x = (0, \ldots, 0)$ is recurrent[3], our model is unichain. Recall that since conditions 1, 3 and 4 from Chapter 2 are satisfied, value iteration will give us the ϵ-optimal solution.

Remark 4. *Optimality of the MDP and the 'power of two'*

In Figure 3.1, we compare fixed SQ(d) policies with the MDP, that chooses d dynamically. The optimality of the MDP is illustrated. To make the comparison fair, the cost functions of the Markov reward processes that model the fixed SQ(d) algorithms, have been adapted, such that also the (fixed) communication costs are taken into account.
In Figure 3.2 the power of 2 is illustrated.

[3]Indeed, for every state y, there exists a path to x. The simplest path would be the one with only departures.

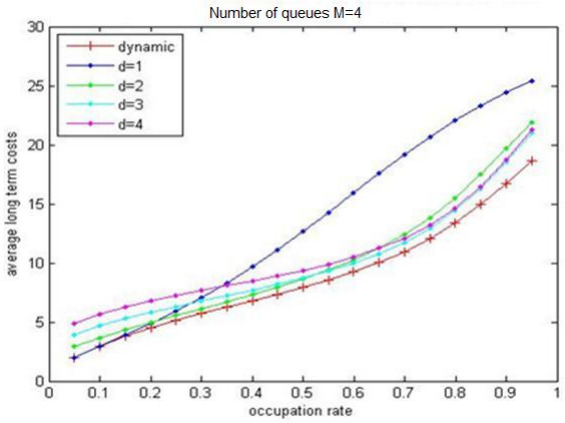


Figure 3.1: The MDP that chooses d dynamically outperforms the fixed SQ(d) algorithms. Here we have taken $f_1(x_M) = x_M$ and $f_2(d) = d$, but this result holds in greater generality.

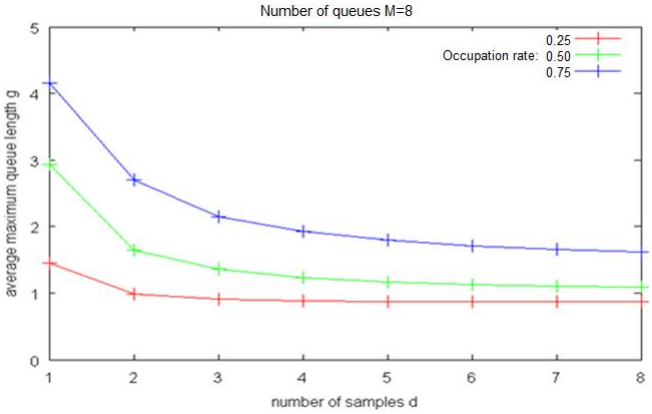


Figure 3.2: Illustration of the 'power of two' for different occupation rates and $M = 8$. We have taken $f_1(x_M) = x_M$ and $f_2(d) = d$, but the idea that the biggest gain in performance is when going from $d = 1$ to $d = 2$, holds in greater generality.

Chapter 4

Structural properties of the relative value function

In this section we will investigate structural properties (such as increasingness) of the relative value function V for the the Dynamic SQ(d) model. In general, once such properties are proven, this can lead to characterization of the optimal policy (see for instance [25]). If such a characterization (e.g., a threshold policy) has been proven, this might in turn lead to a faster execution of the value iteration algorithm, since one can, instead of optimizing over all possible policies, restrict optimization to only those policies that have the same characteristics as the optimal policy.
Secondly, note that the value iteration algorithm attempts to find the global optimum of the relative value function V. Recall that for a convex function it holds that a local optimum is also a global optimum. Therefore, by showing convexity and exploiting this property, it might be possible to develop a faster algorithm to solve the optimality equation.
The two above mentioned arguments are the main reason that structural properties are investigated. In our case, we expect V to be convex and non-decreasing (see Figure 4.1). We have managed to prove non-decreasingness of this relative value function; its proof you find below. Unfortunately, we have not yet been able to prove convexity using standard methods. Hence, either it is not convex anyway, or new techniques have to be developed to prove convexity of V for our model. The attempts that we have made in this regard can be found in Appendix A.1.

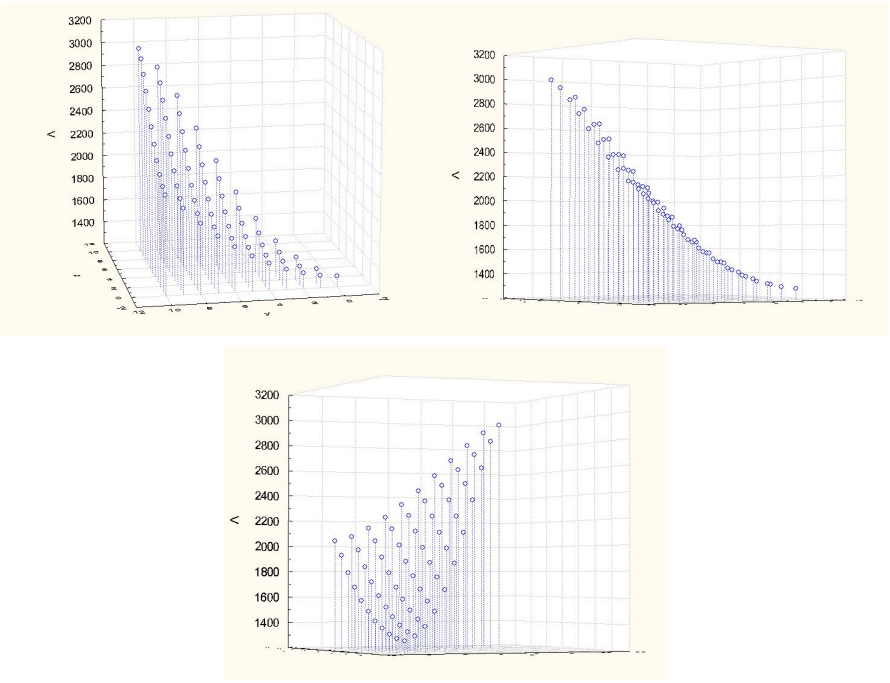


Figure 4.1: Convexity of the relative value function V in the three dimensional case, where we truncated the state space at $K = 10$. At the boundary, it does not all look convex, but we expect this to be due to boundary effects as a consequence of truncating the state space. That is, if we take $K = 15$ instead, it looks all convex around the value 10.

4.1 Prepatory lemmas and notation

Before proving non-decreasingness, we will give some prepatory lemmas and introduce notation. Throughout this section we assume that x is sorted, that is, $\sigma(x) = x$, with σ the sorting algorithm, used in for instance Equation (3.15). The statement in the first lemma is very obvious, but good to establish nevertheless.

Lemma 4.1.1.
a) *The mapping $x \mapsto \sigma(x + e_j)$ is* ***non-decreasing*** *for all $j \in \{1, \dots, M\}$. That is, for every $j \in \{1, \dots, M\}$ there is a $k^* \in \{j, \dots, M\}$ such that $(\sigma(x + e_j))_{k^*} > x_{k^*}$ and for all $k \in \{1, \dots, M\}, k \neq k^*$, it holds that $(\sigma(x + e_j))_k = x_k$ for this j.*
b) *Similarly, the mapping $x \mapsto \sigma(x - e_j)$ is* ***non-increasing*** *for all $j \in \{1, \dots, M\}$. That is, for every $j \in \{1, \dots, M\}$ there exists a $k^{**} \in \{1, \dots, j\}$ such that $(\sigma(x - e_j))_{k^{**}} < x_{k^{**}}$ and $(\sigma(x - e_j))_k = x_k$ for all $k \in \{1, \dots, M\}, k \neq k^{**}$.*

In other words, adding or subtracting e_j to x for a particular $j \in \{1, \dots, M\}$ only changes (increases and decreases, respectively) x, after sorting, in one position (in position k^* and k^{**}, respectively): $\sigma(x + e_j) = x + e_{k^*}$ and $\sigma(x - e_j) = x - e_{k^{**}}$.

Proof. We will only show the first case, as the second case goes similarly.
First note that $\sigma(x + e_M) = x + e_M$, hence for $j = M$ we have $k^* = M$. Now fix $j \in \{1, \dots, M-1\}$. Note that $\sigma(x + e_j) \neq x + e_j$ if and only if $x_j = x_{j+1}$. Now we distinghuish two cases.
Case 1: Suppose $x_j < x_{j+1}$, then $k^* = j$, since $(\sigma(x+e_j))_j = (x+e_j)_j = x_j + 1 > x_j$ and for all $k \neq j$ it holds that $(\sigma(x + e_j))_k = (x + e_j)_k = x_k$.
Case 2: Suppose $x_j = x_{j+1} = \ldots = x_m$ for some $j < m \leq M$. Then, by definition of the sorting function σ, it follows that $\sigma(x + e_j) = x + e_m$ and thus

$$(\sigma(x + e_j))_k = \begin{cases} x_k + 1 & \text{if } k = m, \\ x_k & \text{otherwise.} \end{cases}$$

So in this case $k^* = m$. □

In the proofs that follow later, for fixed j, we will have to make use of the following variables constantly. For notational convenience we already define them here.

Definition 5. *Given x, we define, for fixed j, the variables $\overline{j}, \overline{x}^j, \underline{j}$ and $\underline{x}^j$ as follows. By Lemma 4.1.1 we have that*

$$\sigma(x + e_j) = x + e_{\overline{j}},$$

for $\overline{j}$ with $j \leq \overline{j} \leq M$, the maximum index such that $x_{\overline{j}} = x_j$ (i.e., $\overline{j}$ is the k^* of Lemma 4.1.1 here). Define $\overline{x}^j := x + e_{\overline{j}}$ and note that it satisfies $\sigma(\overline{x}^j) = \overline{x}^j$, i.e., $\overline{x}^j$ is ordered.
Similarly,

$$\sigma(x - e_j) = x - e_{\underline{j}},$$

for $\underline{j}$ with $1 \leq \underline{j} \leq j$ the minimum index such that $x_{\underline{j}} = x_j$ (i.e., $\underline{j}$ is the k^{**} of Lemma 4.1.1 here). Also $\underline{x}^j := x - e_{\underline{j}}$ is ordered.
Obviously, $\underline{j} \leq j \leq \overline{j}$. Note that for $\sigma(x - e_j)$ to exist, we must assume that $x_j \geq 1$ and hence also $x_{\overline{j}} \geq 1$ and $x_{\underline{j}} \geq 1$ holds, since $x_{\underline{j}} = x_j = x_{\overline{j}}$.

The variables involved in Lemma 4.1.1 and Definition 5 above are illustrated in Figures 4.2 and 4.3.

The following lemma is explicitly needed in the proof of non-decreasingness of the relative value function (Theorem 4.2.1).

Lemma 4.1.2. *Fix some $n \geq 0$. Given that*

$$V_n(\sigma(x + e_j)) - V_n(x) \geq 0 \tag{4.1}$$

for all $x \in \mathcal{X}$, with $j = 1, \ldots, M$, then

$$V_n(\sigma(\sigma(x + e_j) - e_i)) - V_n(\sigma(x - e_i)) \geq 0 \tag{4.2}$$

and

$$V_n(\sigma(\sigma(x + e_j) + e_i)) - V_n(\sigma(x + e_i)) \geq 0 \tag{4.3}$$

for all $x \in \mathcal{X}$, with $i, j \in \{1, \ldots, M\}$.

Proof. We will only prove (4.2) as the proof of (4.3) is similar. Fix i and j. Basically, we have to show that the left hand side of (4.2) is equal to

$$V_n(\sigma(x^* + e_h)) - V_n(x^*), \tag{4.4}$$

for some $x^* \in \mathcal{X}$ and $h \in \{1, \ldots, M\}$, because then we can use (4.1).
We will assume that indices l exist with $x_l = x_j \pm 1$. In particular, we will assume that $1 < j < M$. If this is not the case, the proof will only become easier.
Define, $x^* := \sigma(x - e_i)$. By Lemma 4.1.1 we know that there exists an $\underline{i}$ with $1 \leq \underline{i} \leq i$ such that $\sigma(x - e_i) = x - e_{\underline{i}}$ (i.e., we take $x^* := \underline{x}^i$). Now it remains to show that $\sigma(\sigma(x + e_j) - e_i) = \sigma(x^* + e_h)$ for some $h \in \{1, \ldots, M\}$. Recall from Definition 5 the variable $\overline{x}^j = \sigma(x + e_j) = x + e_{\overline{j}}$. Now we have two cases.

Case 1: $\overline{x}^j_{\overline{j}} = \overline{x}^j_M$
Then

$$\begin{aligned}
\sigma(\sigma(x + e_j) - e_i) &= \sigma(x + e_{\overline{j}} - e_i) \\
&= \begin{cases} x + e_{\overline{j}} - e_{\overline{j}} & \text{if } i \geq \overline{j} \\ x + e_{\overline{j}} - e_{\underline{i}} & \text{if } i < \overline{j} \end{cases} \\
&= \begin{cases} x & \text{if } i \geq \overline{j} \\ x - e_{\underline{i}} + e_{\overline{j}} & \text{if } i < \overline{j} \end{cases} \\
&= \begin{cases} \sigma(x^* + e_j) & \text{if } i > \overline{j} \\ \sigma(x^* + e_{\underline{i}}) & \text{if } i = \overline{j} \\ \sigma(x^* + e_j) & \text{if } \underline{j} \leq i < \overline{j}, \underline{j} < j \\ \sigma(x^* + e_{j+1}) & \text{if } \underline{j} \leq i < \overline{j}, \underline{j} = j \\ \sigma(x^* + e_j) & \text{if } i < \underline{j}. \end{cases}
\end{aligned} \tag{4.5}$$

In the second equality, we use for the case $i < \overline{j}$ that the k^{**} from Lemma 4.1.1 is the same (namely $\underline{i}$) for both x and $x + e_{\overline{j}}$ when subtracting e_i.
The last equality follows after identifying that for $i < \overline{j}$, three cases arise. In cases $\underline{j} \leq i < \overline{j}$, $\underline{j} < j$ or $i < \underline{j}$ we have that the k^* from Lemma 4.1.1 is the same (namely $\overline{j}$) for both x and x^* when adding e_j. In case $\underline{j} \leq i < \overline{j}$, $\underline{j} = j$ however, this is not so. In order to get $\overline{j}$ as the k^* for x^* we should add e_{j+1} instead of e_j (note that in this case $x_j = x_{j+1}$ because $j < \overline{j}$).
For the case $i > \overline{j}$ one has to recognize that $\sigma(x^*+e_j) = \sigma(x-e_{\underline{i}}+e_j) = x-e_{\underline{i}}+e_{\underline{i}} = x$, where the second equality follows because the k^* of Lemma 4.1.1 is $\underline{i}$ for state $x - e_{\underline{i}}$ when adding e_j. We could also not have split up this case from the case $i = \overline{j}$, but we have done this nevertheless for later use in the proof of Lemma A.1.1.
So for the cases $i > \overline{j}$ or $\underline{j} \leq i < \overline{j}$, $\underline{j} < j$ or $i < \underline{j}$ we have that $h = j$ gives us (4.4). For the case $i = \overline{j}$ we take $h = \underline{i}$ to get (4.4) and for the case $\underline{j} \leq i < \overline{j}$, $\underline{j} = j$ we take $h = j + 1$.
The case where $i > \overline{j}$, is illustrated in Figure 4.4.

Case 2: $\overline{x}^j_{\overline{j}} < \overline{x}^j_M$
In this case there exists an $m > \overline{j}$ such that $\sigma(\overline{x}^j + e_m) = \overline{x}^j + e_m$. Compared to the first case, not much changes; only for $i > m$ we should again recognize (as for the case $i < \underline{j}$) that the k^{**} from Lemma 4.1.1 is the same (namely $\underline{i}$) for both x and $x + e_{\overline{j}}$ when subtracting e_i. So we get

$$\sigma(\sigma(x + e_j) - e_i) = \sigma(x + e_{\overline{j}} - e_i)$$
$$= \begin{cases} \sigma(x^* + e_j) & \text{if } \overline{j} < i \leq m \\ \sigma(x^* + e_{\underline{i}}) & \text{if } i = \overline{j} \\ \sigma(x^* + e_j) & \text{if } \underline{j} \leq i < \overline{j}, \underline{j} < j \\ \sigma(x^* + e_{j+1}) & \text{if } \underline{j} \leq i < \overline{j}, \underline{j} = j \\ \sigma(x^* + e_j) & \text{if } i < \underline{j} \text{ and } i > m. \end{cases}$$

and (4.4) follows similarly as in Case 1. □

Note that since in both cases the value of h for all i except for $i = \overline{j}$ and $\underline{j} \leq i < \overline{j}, \underline{j} = j$ are the same, we could have taken them together (and we needed not have distinguished between Case 1 and Case 2 either), but because there are differences in analysis in order to come to this h, we presented them distinctly.

In order to present the proofs of Theorem 4.2.1 below and Conjecture A.1.1 in the appendix more compactly, we have the following definition.

Definition 6. *For $x \in \mathcal{X}$, $d \in \mathcal{D}$, $n \geq 0$ and given $M > 1$, we define*

$$T_d^n(x) := f_2(d) + \lambda \sum_{i=1}^{M-d+1} k(M,i,d) V_n(\sigma(x+e_i)),$$

with $k(M,i,d) = d\frac{(M-i)!(M-d)!}{(M-i-d+1)!M!}$.

4.2 Non-decreasingness

The following theorem establishes the property of non-decreasingness of the relative value function V.

Theorem 4.2.1. *For non-decreasing f_1, it holds that V is non-decreasing, i.e., $V(\sigma(x+e_j)) - V(x) \geq 0$ for all $x \in \mathcal{X}$ and $j = 1, \ldots, M$.*

Proof. We will prove this by induction on n in V_n. Take $V_0(x) = 0$ for all x. Then obviously, $V_0(x)$ is non-decreasing for all x. Now assume that $V_n(x)$ is non-decreasing, i.e., $V_n(\sigma(x+e_j)) - V_n(x) \geq 0$. Then, for $n+1$ we have the following.

$$\begin{aligned} V_{n+1}(\sigma(x+e_j)) - V_{n+1}(x) &= \underbrace{f_1((\sigma(x+e_j))_M) - f_1(x_M)}_{A} \\ &+ \mu \left(\underbrace{\sum_{i=1}^{M} \mathbb{I}\{(\sigma(x+e_j))_i > 0\} V_n(\sigma(\sigma(x+e_j) - e_i)) - \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} V_n(\sigma(x-e_i))}_{B} \right) \\ &+ \underbrace{\left[1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{(\sigma(x+e_j))_i > 0\})\right] V_n(\sigma(x+e_j)) - \left[1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\})\right] V_n(x)}_{C} \\ &+ \underbrace{\min_{d \in \mathcal{D}} T_d^n(\sigma(x+e_j)) - \min_{d \in \mathcal{D}} T_d^n(x)}_{D}. \end{aligned} \tag{4.6}$$

Note that for A it holds that for f_1 non-decreasing

$$f_1((\sigma(x+e_j))_M) - f_1(x_M) \geq 0,$$

since $(\sigma(x+e_j))_M \geq x_M$ by Lemma 4.1.1.
For B and C we distinguish two cases. Recall from Definition 5 that $\sigma(x+e_j) = x+e_{\bar{j}}$.

Case 1: $x_{\bar{j}} > 0$
Looking at B it holds that $(\sigma(x + e_j))_i \geq x_i$ and hence

$$\mathbb{I}\{(\sigma(x + e_j))_i > 0\} \geq \mathbb{I}\{x_i > 0\},$$

implying

$$B \geq \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} \left[V_n(\sigma(\sigma(x + e_j) - e_i)) - V_n(\sigma(x - e_i))\right].$$

By the induction hypothesis and Lemma 4.1.2 this is larger than or equal to 0.
For C we notice that since $x_{\bar{j}} > 0$ it holds that

$$\mathbb{I}\{x_i > 0\} = \mathbb{I}\{(\sigma(x + e_j))_i > 0\}.$$

Hence

$$C = \left[1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{(\sigma(x + e_j))_i > 0\}\right] \left(V_n(\sigma(x + e_j)) - V_n(x)\right),$$

which is again larger than or equal to 0 because of the induction hypothesis and the fact that

$$\left[1 - \left(\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{(\sigma(x + e_j))_i > 0\}\right)\right] \geq 0.$$

Case 2: $x_{\bar{j}} = 0$
For B, we could in principle use the same argument as for Case 1. However, we need to approach it differently because we will need to compensate a term from C. We write

$$\begin{aligned} B \geq & \sum_{i \neq \bar{j}} \mathbb{I}\{x_i > 0\} \left[V_n(\sigma(\sigma(x + e_j) - e_i)) - V_n(\sigma(x - e_i))\right] \\ & + \underbrace{\mathbb{I}\{x_{\bar{j}} + 1 > 0\}}_{=1} V_n(\underbrace{\sigma(\sigma(x + e_j) - e_{\bar{j}})}_{=x}) - \underbrace{\mathbb{I}\{x_{\bar{j}} > 0\}}_{=0} V_n(\sigma(x - e_{\bar{j}})) \\ \geq & V_n(x), \end{aligned}$$

where we have again used the induction hypothesis and Lemma 4.1.2. For C we can write in this case

$$\begin{aligned} C = &\left[1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{(\sigma(x+e_j))_i > 0\})\right](V_n(\sigma(x+e_j)) - V_n(x)) \\ &-\mu \underbrace{\mathbb{I}\{x_{\bar{j}} = 0\}}_{=1} V_n(x) \\ &\geq -\mu V_n(x), \end{aligned}$$

where we have used the induction hypothesis and Lemma 4.1.2 in the last inequality. Taking the left-over terms of B and C together gives 0, hence also $\mu B + C \geq 0$ in this case.

With regard to D, note that for

$$d^* \in \underset{d \in \mathcal{D}}{\operatorname{argmin}}\{T_d^n(\sigma(x+e_j))\},$$

it holds that

$$\min_{d \in \mathcal{D}} T_d^n(x) \leq T_{d^*}^n(x)$$

and thus

$$-\min_{d \in \mathcal{D}} T_d^n(x) \geq -T_{d^*}^n(x).$$

Hence

$$\begin{aligned} D &\geq T_{d^*}^n(\sigma(x+e_j)) - T_{d^*}^n(x) \\ &= (f_2(d^*) - f_2(d^*)) + \lambda \left[\sum_{i=1}^{M-d^*+1} k(M,i,d^*)\left(V_n(\sigma(\sigma(x+e_j)+e_i)) - V_n(\sigma(x+e_i))\right)\right] \\ &\geq 0, \end{aligned}$$

by the induction hypothesis and Lemma 4.1.2. Taking all this together, it follows that

$$V_{n+1}(\sigma(x+e_j)) - V_{n+1}(x) \geq 0,$$

which is what we wanted to prove, since by taking $n \to \infty$ the result follows:

$$\begin{aligned} 0 &\leq \lim_{n\to\infty} V_{n+1}(\sigma(x+e_j)) - V_{n+1}(x) \\ &= \lim_{n\to\infty} [V_{n+1}(\sigma(x+e_j)) - g(n+1)] - [V_{n+1}(x) - g(n+1)] \\ &= \lim_{n\to\infty} [V_{n+1}(\sigma(x+e_j)) - g(n+1)] - \lim_{n\to\infty} [V_{n+1}(x) - g(n+1)] \\ &= V(\sigma(x+e_j)) - V(x), \end{aligned}$$

where we have used (2.8) in the last equality. □

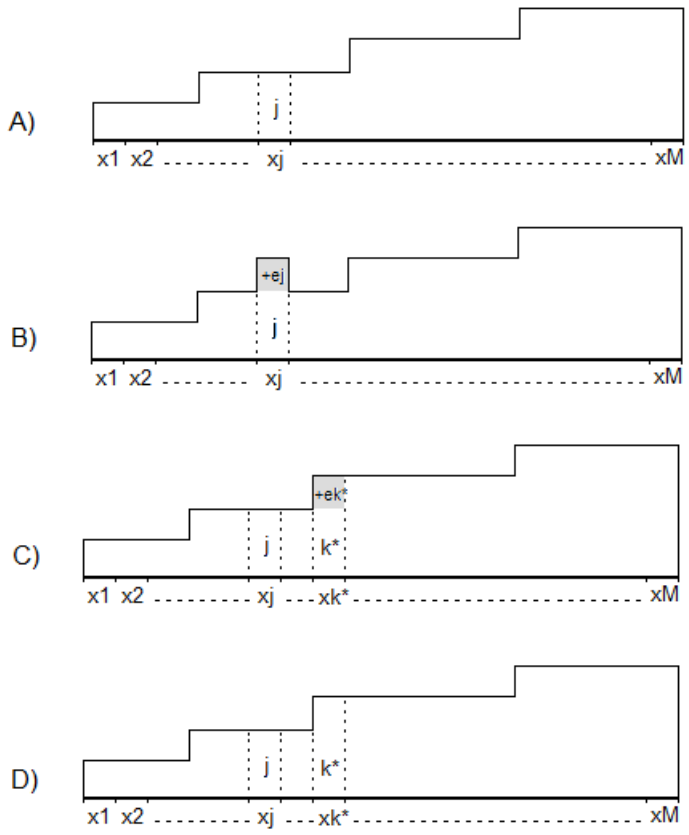


Figure 4.2: Illustration of some of the variables involved in Lemma 4.1.1 and Definition 5. In A) some $x \in \mathcal{X}$ is depicted. In B) we see $x + e_j$ and in C) we see how k^* is defined. Here $\overline{j} = k^*$. Finally in D) we find $\overline{x}^j := \sigma(x + e_j) = x + k^*$.

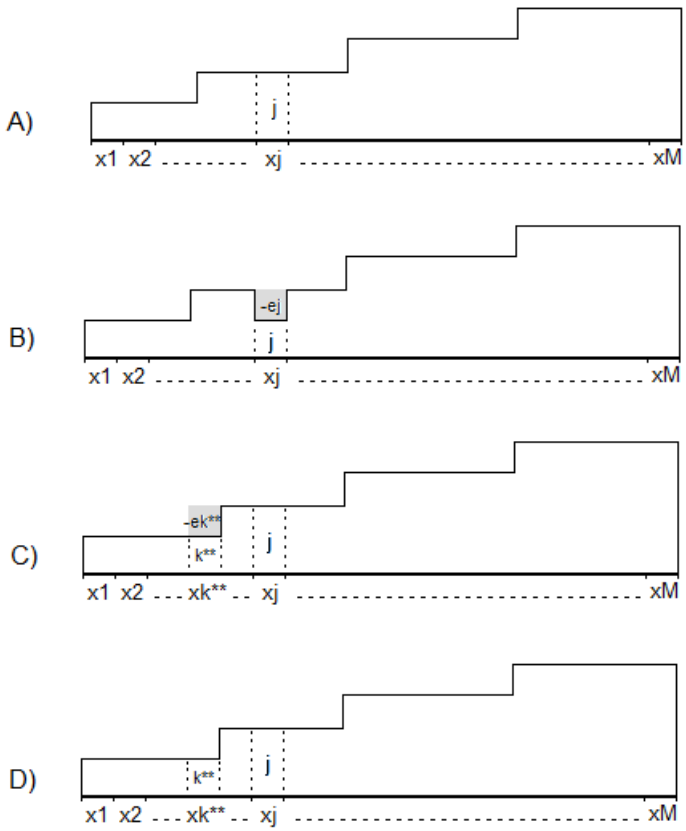


Figure 4.3: Illustration of some of the variables involved in Lemma 4.1.1 and Definition 5. In A) some $x \in \mathcal{X}$ is depicted. In B) we see $x - e_j$ and in C) we see how k^{**} is defined. Here $\underline{j} = k^{**}$. Finally in D) we find $\underline{x}^j := \sigma(x - e_j) = x - k^{**}$.

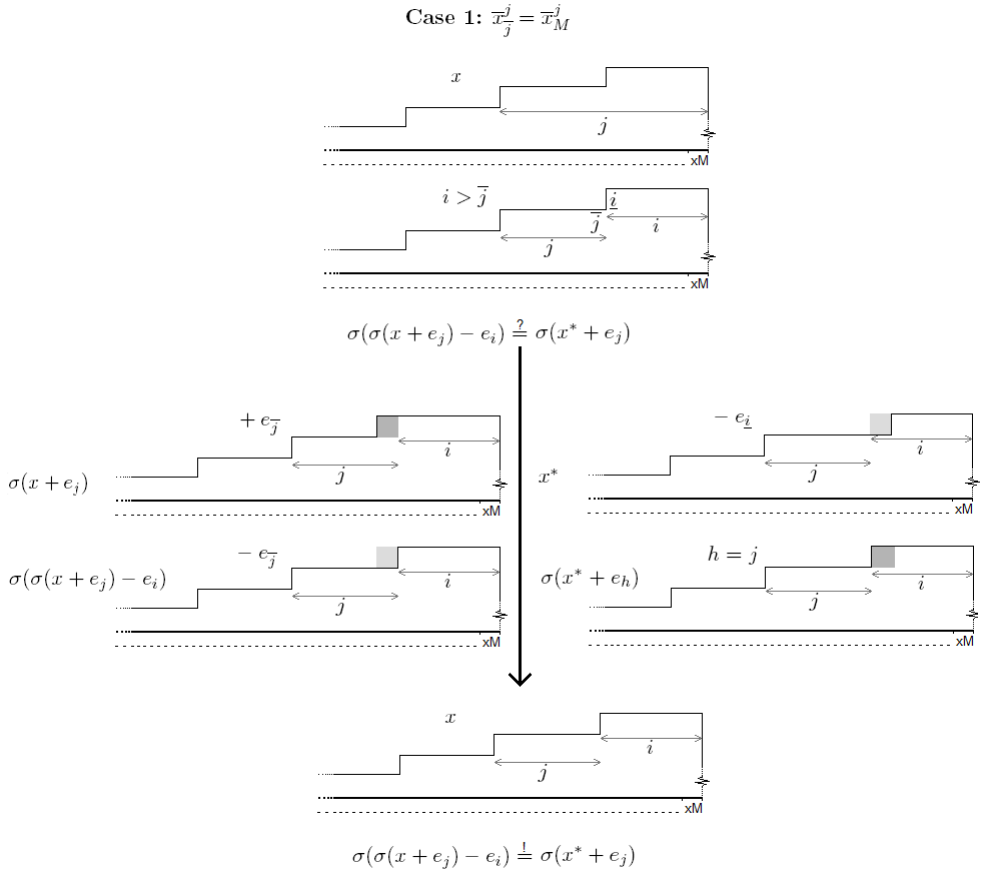


Figure 4.4: Illustration of the proof that for the case $\overline{x}^j_{\overline{j}} = \overline{x}^j_M$ and $i > \overline{j}$, it holds that there exists an $h \in \{1, \ldots, M\}$ such that $\sigma(\sigma(x + e_j) - e_i) = \sigma(x^* + e_h)$, with $x^* = x - e_{\underline{i}}$. In this case $h = j$ suffices.

Chapter 5

Heuristics and results

In the previous chapters we have throughout assumed that we had complete information on the state of the system. That is, at all times we knew exactly the number of customers in every subsystem. Consequently, we knew at all times the maximum number of customers over all subsystems and hence the associated costs. An MDP which has complete information, is called a Markov decision process with full information. An example is the MDP we formulated in Chapter 3 for the dynamic SQ(d) policy. As stated in the introduction, this is not the case we are interested in. That is, we want to investigate the case where we do not have complete state information, but we can only use what we know from previous decisions and samples. These kind of MDPs are also called Markov decision processes with partial information. In Section 5.1 we formulate such an MDP for the dynamic SQ(d) policy and show how to transform this MDP with partial information into an equivalent MDP with full information. Note, however, that this MDP with full information is not the same as the one we formulated in Chapter 3, since there the states really represented the number of jobs in each subsystem, whereas here, the states will be a *distribution* over the number of jobs in each subsystem. Simply stated, this distribution represents our 'guess' of how many jobs are present in each subsystem.
However, as mentioned in Section 2.5, Markov decision models suffer from the so-called 'curse of dimensionality' and are not scalable[1] in practice. For Markov decision processes with partial information this is even worse, as will become clear below.
In Section 5.2 we try to deal with this scalability problem in different ways and finally come up with a scalable algorithm that outperforms SQ(2) and JSQ for certain scenarios in the case without complete information.

[1]Recall that with not being 'scalable', we mean here that as we increase the number of subsystems, the corresponding computations will become infeasible.

5.1 Partial information model

In this section we model the dynamic SQ(d) algorithm –in the case where we do not have complete information– as a Markov decision proces with partial information. The key to formulating an MDP with partial information is that we have to come up with an appropriate state space. To this end, we distinguish between the state of the model (with state space $\mathcal{X}$)and the state of the algorithm (with *information state space* $\mathcal{P}$) that represents the information we have. The information state is actually a distribution on $\mathcal{X}$ (it is also called the *belief state*, since this distribution is our belief of what state the model is in). Next to this, we have an *observation space* $\mathcal{Z}_d$, which in our case represents the exact number of jobs in the d subsystems that are sampled. The observation space depends on d, the decision, which stands for the number of subsystems sampled. Thus,the decision space is the same as before, i.e., $\mathcal{D} = \{1, \ldots, M\}$.
Remark that the information state holds all information on all observations up to the current time. Thus, it is observable (meaning, in a sense, that we have full information), it only depends on the observations, and the Markov property holds. Hence, in our setting, the information state can theoretically be used to solve the problem: Starting with some (well-chosen) prior distribution, the information state is updated every time unit in a Bayesian manner. By using the value iteration algorithm with $\mathcal{P}$ as state space, we can compute the optimal policy, given the initial distribution. This observation is formalized in Theorem 4.1 of [10], which states that there exists a full information model that is equivalent to every partial information model (i.e., the same optimal costs and an optimal policy in the full information model is also optimal in the partial information model), under some conditions that hold in our case.
Practically, however, note that for state space $\mathcal{X}$, the information state is given by $\mathcal{P} = [0, 1]^{|\mathcal{X}|}$, which is an $|\mathcal{X}|$-dimensional space, with each information state variable a probability. In particular, $\mathcal{P}$ is not countable and we would certainly have to discretize the state space to make computations possible. However, where discretization of the interval $[0, 1]$ could make computations possible for small state spaces, for our problem with multidimensional state variables this is infeasible.
Despite these practical challenges, let us give the partial information model of our problem anyway. As mentioned, we will to this end use Theorem 4.1 of [10], from which the following general framework follows. Given are a state space $\mathcal{X}$, a decision space $\mathcal{D}$, a transition probability function p and a cost function c. With the observation space $\mathcal{Z}_d$ we define the so-called observation probabilities $q(x, d, z)$ as the probability of observing z in state x when taking action d. We will now define

transition probabilities $\tilde{p}(u, d, v)$ and cost function $\tilde{c}(u, d)$, with $u, v \in \mathcal{P}$ and $d \in \mathcal{D}$, for the Markov decision process with $\mathcal{P}$ as state space and $\mathcal{D}$ as decision space. Given a $d \in \mathcal{D}$ and a $z \in \mathcal{Z}_d$, the transition probabilities are given by

$$\tilde{p}(u, d, v) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{X}} u(x) p(x, d, y) q(y, d, z),$$

for v such that

$$v(y) = \frac{\sum_{x \in \mathcal{X}} u(x) p(x, d, y) q(y, d, z)}{\sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} u(x) p(x, d, x') q(x', d, z)},$$

and $v(y) = 1$ if the denominator is equal to 0. Define this v as $T_y(u, d, z)$. Note that we have assumed that, for u, different observations z lead to different v. If, for fixed (u, d), there are different observations z and z' such that $T_y(u, d, z) = T_y(u, d, z')$, then we should define $\tilde{p}$ as

$$\tilde{p}(u, d, v) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{X}} u(x) p(x, d, y) q(y, d, z) \mathbb{I}\{T_y(u, d, z) = v\},$$

with, as before, $\mathbb{I}\{.\}$ the indicator function.

In order to complete our particular full information model for the dynamic SQ(d) Markov decision process with partial information, we should give the following specifications to the above.

$$\begin{aligned} \text{The state space } \mathcal{X} &:= \left(\mathbb{N}_0^K\right)^M, \\ \text{the decision space } \mathcal{D} &:= \{1, \ldots, M\}, \\ \text{the observation space } \mathcal{Z}_d &:= \left(\mathbb{N}_0^K \times \mathcal{D}\right)^d, \\ \text{the information space } \mathcal{P} &:= [0, 1]^{|\mathcal{X}|} \text{ and} \\ \text{the observation probabilities } q(x, d, z) &:= \frac{d!}{M(M-1)\cdots(M-d+1)}, \end{aligned} \tag{5.1}$$

where $\mathbb{N}_0^K := \{0, 1, \ldots, K\}$. The observation space $\mathcal{Z}_d$ is of the form given, because for every sample we want to know the number of jobs ($\in \mathbb{N}_0^K$) in the subsystem ($\in \mathcal{D}$) sampled. For the observation probabilities, recall that $q(x, d, z)$ is the probability of observing $z \in \mathcal{Z}_d$, given that the system is in $x \in \mathcal{X}$ and $d \in \mathcal{D}$ subsystems are sampled. There are $\binom{M}{d} = \frac{M!}{(M-d)!d!}$ possibilities to choose d subsystems out of M, so the probability to choose one particular combination is

$$q(x, d, z) = \frac{1}{\binom{M}{d}} = \frac{(M-d)!d!}{M!} = \frac{d!}{M(M-1)\cdots(M-d+1)}.$$

5.2 Heuristic approach

As remarked in the previous section, it is practically infeasible to compute an optimal policy for the full information model (5.1) above. Even after discretization, the sheer dimensionality of our framework prohibits this. We could circumvent these computational problems to some extent by choosing a heuristic approach. To this end, we will decouple the computations into

Phase A: The approximation of the model state;

Phase B: The determination of a good corresponding policy.

We will first focus on Phase A, where we will treat the computation of the information state. With this information state, we will be able to approximate the state of the model. The idea of the heuristic is that given this approximated state of the model (Phase A), it uses the MDP from Chapter 3 to come to a hopefully good policy for that approximated state (Phase B). This decoupling mechanism will be the first approximation step. Since the resulting heuristic will not be scalable either due to the complexity of the MDP it uses, in the second approximation step we focus on lowering the computation time in Phase B. To do so, we will approximate the relative value function V of the MDP involved. This results in a heuristic that is scalable up to at least 30 subsystems.
In the remaining part of this section, in our search for a well-performing policy in Phase B given the outcomes of Phase A, we will call the resulting policy simply the "corresponding policy". It should be clear that we want this corresponding policy to be as good as possible.
Contrary to what we did previously, here we do not sort our model states. This is not needed because we will not use the transition probabilities defined in Chapter 3. Moreover, the generalization to different service rates will be more natural later on.

5.2.1 Approximation step 1: Estimating the state variable

In this section, we will explain the decoupling mechanism, which is a first step to a scalable heuristic. Initially, our focus is on Phase A where we concentrate on the information space, that basically is a distribution over all possible model states. Secondly, we will give different options of how to use the resulting distribution from Phase A in Phase B.

Phase A
Assume we have M subsystems with buffer length K, i.e., the total number of possible jobs waiting in the queue plus the job in service is maximally[2] K. We define two M-dimensional vectors $\tau = (\tau_1, \ldots, \tau_M)$ and $s = (s_1, \ldots, s_M)$ to be able to keep track of the distribution we are interested in. τ_i denotes the time since subsystem i was last sampled and s_i denotes the number of jobs in subsystem i at the moment it was last sampled. Instead of keeping a distribution over all possible states directly, we will keep M distributions over the number of jobs for each subsystem. This second option is more general and allows for more different options when it comes to calculating a policy in Phase B. In this case, given vectors τ and s, at an arrival we could calculate per subsystem the probability that there are $j = 0, 1, \ldots, K$ jobs in the subsystem using the following observation.
Since we assume that we start with an empty system, and because τ and s are updated at every arrival using the information we get by sampling subsystems, the biggest challenge is to estimate how many jobs have left since the last time a subsystem was sampled. Due to the exponential service times with rate μ, we can think of service events being given in terms of a Poisson process with rate μ, such that an event in the Poisson process corresponds to a job having finished service if the subsystem is nonempty and is just dummy otherwise. Let $A(t)$ be the number of Poisson events in the interval $(0, t]$. As we know from the counting process of a Poisson process, $A(t)$ is Poisson distributed and

$$q_k(t) := P(A(t) = k) = e^{-\mu t} \frac{(\mu t)^k}{k!}.$$

So given $\tau = (\tau_1, \ldots, \tau_M)$ and $s = (s_1, \ldots, s_M)$, at an arrival we have that in subsystem i, $s_i - k$ jobs are present with probability $q_k(\tau_i)$, if $s_i - k > 0$ and 0 jobs are present otherwise. The probability that 0 jobs are present, happens with probability $1 - \sum_{j=0}^{s_i - 1} q_j(\tau_i)$.
Using this technique and the resulting distribution per subsystem, we can

I Calculate the most likely x, denoted x^l;

II Calculate the expected x, denoted x^e,

where x is the model state we do not know, but want to estimate in some way. Calculating x^l is done by taking per subsystem the number of jobs that has highest probability. On the other hand, calculating x^e is done by taking the expected

[2]In Chapter 2, K was called the truncation level. We need this here because we use the optimal policy of the MDP from Chapter 3, which has been calculated using a truncation level K.

value of the number of jobs per subsystem and rounding each to the nearest integer[3]. Both method I and II are illustrated in Figure 5.1. Furthermore, from the M distributions per subsystem, we could also compute the distribution over all possible model states, since the distributions per subsystem are independent, given vectors τ and s. That is, the probability that according to this distribution, the model state x is some particular y, denoted $P(x = y)$, can be computed as follows: $P(x = y) = P(x_1 = y_1) \cdots P(x_M = y_M)$, where $P(x_i = y_i)$ is the probability that in subsystem i there are y_i jobs present. Denote this distribution $\mathcal{Q}$. The way distribution $\mathcal{Q}$ can be computed, is also illustrated in Figure 5.1.

# of jobs	x1	x2
0	0.1	0.2
1	0.2	0.3
2	0.3	0.5
3	0.4	0
4	0	0
...	0	0
...	0	0

Figure 5.1: In this figure you find an illustration of methods I and II mentioned in the text. We have two subsystems in this example and for both subsystems a distribution over the number of jobs in the subsystem is given. Method I takes for both subsystems the most likely number of jobs, which then gives the most likely x. In this case this would result in 3 jobs in subsystem 1, and 2 jobs in subsystem 2, since these appear with highest probability in their respective columns, i.e., $x^l = (3, 2)$. Method II calculates the expected number of jobs in both subsystems, which gives the expected x. So in this case this results in $[0 + 0.2 + 0.6 + 1.2] = [2] = 2$ jobs in subsystem 1, and $[0 + 0.3 + 1] = [1.3] = 1$ job in subsystem 2, where [.] is the rounding function. So, $x^e = (2, 1)$. Note that $x^l \neq x^e$ in this case, i.e., these methods may give a different result. Finally, to illustrate how distribution $\mathcal{Q}$ can be computed, recall that $P(x = y) = P(x_1 = y_1)P(x_2 = y_2)$, hence e.g., $P(x = (3, 0)) = 0.4 * 0.2 = 0.08$ and $P(x = (1, 2)) = 0.2 * 0.5 = 0.1$.

Phase B

Given the distributions that represent the information state from Phase A, we can

[3]One can also round it up or down, but in our tests we did not observe a significant improvement when doing so.

do different things to calculate a corresponding policy. Firstly, we already saw two different ways of how the distribution could be used to come to an estimated x, (see also Figure 5.1). Let this estimated x be denoted $\hat{x}$ (so $\hat{x} = x^l$ or $\hat{x} = x^e$ may hold). To illustrate how we could use $\hat{x}$ to come to a corresponding policy, consider the following.
Let us assume that we consider a system with M subsystems, Poisson arrivals with rate λ and exponential service times with rate μ, equal for all subsystems. Note that we want to run our dynamic SQ(d) policy and want to analyze this system in the case where we do not have complete information. To do so, we would like to use the MDP we formulated in Chapter 3. Recall that if we choose a truncation level K, then this MDP would give us an optimal policy for the case with complete information. That is, the results of the MDP give us a mapping

$$x \mapsto d^*(x), \tag{5.2}$$

where $d^*(x)$ is the optimal number of samples in model state x. However, two things should be dealt with in order to be able to use this mapping. Firstly, the state space of the MDP from Chapter 3 consists of sorted model states x only. That is, $d^*(.)$ is only defined on $\mathcal{X} := \{z \in \mathbb{N}_0^M | z_1 \leq z_2 \leq \cdots \leq z_M\}$. So to be able to use this mapping, we should make sure that $x \in \mathcal{X}$. Secondly, contrary to the setting in which the MDP in Chapter 3 is defined, we are interested in the case where we do not have complete information. Taking the above observations into account and combining $\hat{x}$ with mapping d^* immediately gives us two options for our heuristic:

1. Calculate x^l at every arrival and sample $d^*(\sigma(x^l))$ subsystems; we call this heuristic "*heurlike*".

2. Calculate x^e at every arrival and sample $d^*(\sigma(x^e))$ subsystems; we call this heuristic "*heurexp*".

Remark 5.

The way we have used the MDP from Section 3, in fact comes down to the following. In the corresponding optimality equations (see (3.15)), we see that mapping (5.2) hinges on the expression

$$\min_{d \in \mathcal{D}} \left\{ f_2(d) + \lambda \sum_{i=1}^{M-d+1} k(M,i,d) V(\sigma(x + e_i)) \right\}. \tag{5.3}$$

So basically, calculating $d^*(\hat{x})$ for the above-defined heuristics comes down to calculating

$$\underset{d\in\mathcal{D}}{\operatorname{argmin}} \left\{ f_2(d) + \lambda \sum_{i=1}^{M-d+1} k(M,i,d) V(\sigma(\sigma(\hat{x}) + e_i)) \right\}, \tag{5.4}$$

for $\hat{x} = x^l$ or $\hat{x} = x^e$, respectively. Hence, given V, we have

$$d^*(x) := \underset{d\in\mathcal{D}}{\operatorname{argmin}} \left\{ f_2(d) + \lambda \sum_{i=1}^{M-d+1} k(M,i,d) V(\sigma(x + e_i)) \right\}, \tag{5.5}$$

for $x \in \mathcal{X}$.

Observe that both heuristics do not fully exploit the power of the distribution from Phase A. Indeed, instead of taking $d^*(x_l)$ or $d^*(x_e)$, we could also take $\mathbb{E}[d^*(x)]$, where $\mathbb{E}$ is the expectation with respect to $\mathcal{Q}$:

$$\mathbb{E}[d^*(x)] := \sum_y P(x = y) \underset{d\in\mathcal{D}}{\operatorname{argmin}} \left\{ f_2(d) + \lambda \sum_{i=1}^{M-d+1} k(M,i,d) V(\sigma(\sigma(y) + e_i)) \right\}. \tag{5.6}$$

Note that in the argument of V we had to put $\sigma(y)$ instead of just y, since $y \in \mathcal{X}$ does not necessarily hold. Thus, a third option for our heuristic would be:

3. Calculate $\mathcal{Q}$ at every arrival and sample $\mathbb{E}[d^*(x)]$ subsystems; we call this heuristic "*heurexpvalue*".

We expect this heuristic to outperform the other two, because it takes full advantage of the information available, since it weighs the optimal policy of the MDP with the distribution from phase A.

5.2.2 Results

To sum up, we have got three candidates for our heuristic at the moment:

1. Heuristic "heurlike", that calculates x^l at every arrival and subsequently samples $d^*(\sigma(x^l))$ subsystems;

2. Heuristic "heurexp", that calculates x^e at every arrival and subsequently samples $d^*(\sigma(x^e))$ subsystems;

3. Heuristic "heurexpvalue", that calculates $\mathcal{Q}$ at every arrival and subsequently samples $\mathbb{E}[d^*(x)]$ subsystems; we call this heuristic "heurexpvalue".

It is very interesting to compare the performance of these algorithms with the optimal policy of the MDP from Chapter 3, because this gives an idea as to how far from optimal these heuristics perform. Furthermore, comparing these heuristics with standard algorithms from literature (such as Round Robin, Random, SQ(2) and JSQ), will give us insight in whether choosing d dynamically rather than fixed, will indeed lead to better performance.

First of all, we have to specify a cost function and a truncation level K. In general, we assume that the cost function is of the form $f_1(x_M) + f_2(d)$, with x_M the number of jobs in the subsystem with the highest number of jobs (conform Chapter 3). For our computations we assume throughout this thesis a linear cost function of the form $w_1 x_M + w_2 d$, where we fix $w_2 = 1$ and vary w_1. Since we expect the waiting time due to the queue sizes in the subsystems to be longer than the waiting time due to sampling[4], we vary w_1 only for values such that $w_1 > w_2$.
With regard to the truncation level K, as mentioned in Chapter 2, we have to choose a K such that the difference between g^K and g^{K+1} is small enough, where g^K and g^{K+1} are the long term average cost of the MDP when we use truncation level K and $K+1$, respectively. For $M = 4$ subsystems and $w_1 = 5$, we found taking $K = 11$ sufficient, since then the relative difference, $\frac{g^{K+1} - g^K}{g^K}$, is $< 1\%$ for occupation rates $\rho = 0.7, 0.8$ and $< 2\%$ for $\rho = 0.9$. This is illustrated in Table 5.1. For lower occupation rates the relative difference will only be smaller and even lower values of K would suffice.

Remark 6.

As mentioned earlier, since we use the MDP to calculate a policy, we have to truncate the state space, so that it is finite. In our heuristics however, this truncation level K might be exceeded. In such cases, it may happen that $s_i > K$ for some $i \in \{1, \ldots, M\}$ and we will then take $s_i = K$ instead. This might have consequences for the accuracy of the heuristics, in particular if the truncation level K is chosen too small.

To get an idea of the performance of the three heuristics, let us look at the case with $M = 4$ subsystems and $w_1 = 5$. In the left picture of Figure 5.2 you see the

[4]This idea was supported by Y. Lu, main author of [13], in a conversation we had with her at the IFIP Performance conference in Amsterdam last October.

ρ	g^K	g^{K+1}	$\frac{g^{K+1}-g^K}{g^K}$ (%)
0.7	10.9557	10.9558	9.1277e-004
0.8	13.7529	13.7606	0.0560
0.9	19.6732	19.9457	1.3851

Table 5.1: Comparing g^K with g^{K+1} for various occupation rates ρ.

performance of the heuristics compared to the optimal performance of the MDP. In the right picture we depict the relative difference of the heuristics with the MDP. The relative difference is defined as $\frac{g_h - g_{MDP}}{g_{MDP}}$, where g_h is the long term average cost of the heuristic and g_{MDP} the long term average cost of the MDP. One can see that up to an occupation rate of $\rho = 0.8$, all three heuristics perform within a 10 % bound from optimal. Moreover, it can be seen that from the three heuristics "heurexpvalue" performs best, as expected.

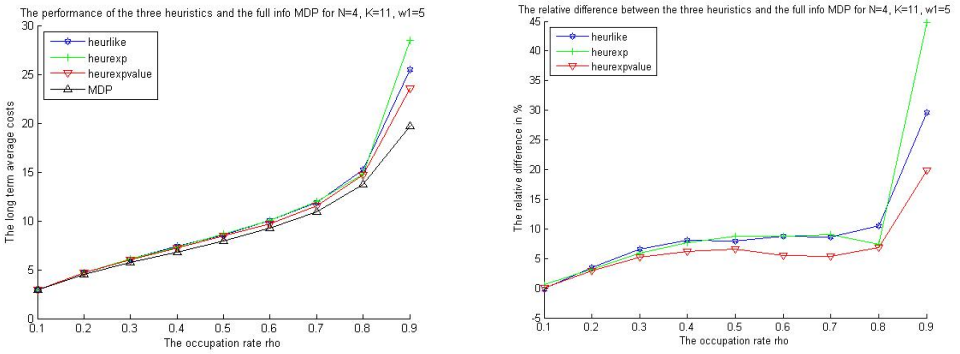


Figure 5.2: Performance of the three heuristics "heurlike", "heurexp" and "heurexpvalue" compared to the performance of the MDP from Chapter 3, for $M = 4$ and $w_1 = 5$. In the picture on the left the performance is given in terms of the long term average costs, whereas in the picture on the right, the relative difference (in %) of the three heuristics with respect to the MDP is depicted.

In Figure 5.3 we compare Round Robin (RR), SQ(1) (=Random), SQ(2) and Join the Shortest Queue (JSQ) with the MDP and see that for medium to high occupation rates, RR and SQ(1) perform very poorly. Since this bad performance also holds for other sets of parameters and since from experience we know that most systems have an occupation rate $\rho \in [0.7, 0.8]$, we will exclude RR and SQ(1) from comparison in the remaining part of this thesis.

Noteworthy is that RR outperforms all other algorithms (also the MDP) for low occupation rates. This does not conflict with the optimality of the MDP however, since the MDP is only optimal within the class of SQ(d)-like policies, to which RR does not belong. Intuitively, it is not hard to reason why RR outperforms the other algorithms for low occupation rates. Indeed, for low occupation rates it is reasonable to assume that every time a job is assigned to a subsystem according to RR, it most likely finds an idle server (so in fact we say that it is reasonable to assume that the service time of a job is generally smaller than M interarrival times). Hence, the lack of communication is only advantegeous here because it does not bring any additional sampling costs, while at the same time there is generally no (or negligible) extra waiting time due to the queue sizes.

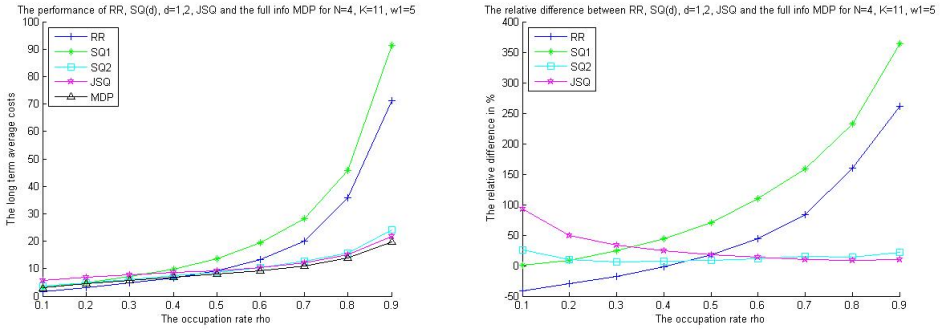


Figure 5.3: Performance of RR, SQ(1), SQ(2), JSQ compared to the MDP from Section 3, for $M = 4$ and $w_1 = 5$. In the picture on the left the performance is given in terms of the long term average costs, whereas in the picture on the right, the relative difference (in %) of the four algorithms with respect to the MDP is depicted.

Finally, we compare our heuristics with SQ(2) and JSQ, as these are the most commonly used SQ(d)-related algorithms used in practice. In Figure 5.4 we see that for these parameters, our heuristics perform fairly well over the whole range of occupation rates, except perhaps for $\rho = 0.9$.
Since we are not interested in systems with only $M = 4$ servers, we will not draw any definite conclusions from the given results in this section. However, these results do give some interesting insights, from which we can learn in our search for a well-performing heuristic for higher dimensional systems. In particular, we find that all three proposed heuristics perform fairly well, but that "heurexpval" performs best, as expected. The heuristics compete with standard algorithms such as SQ(2) and JSQ over the whole range of occupation rates.

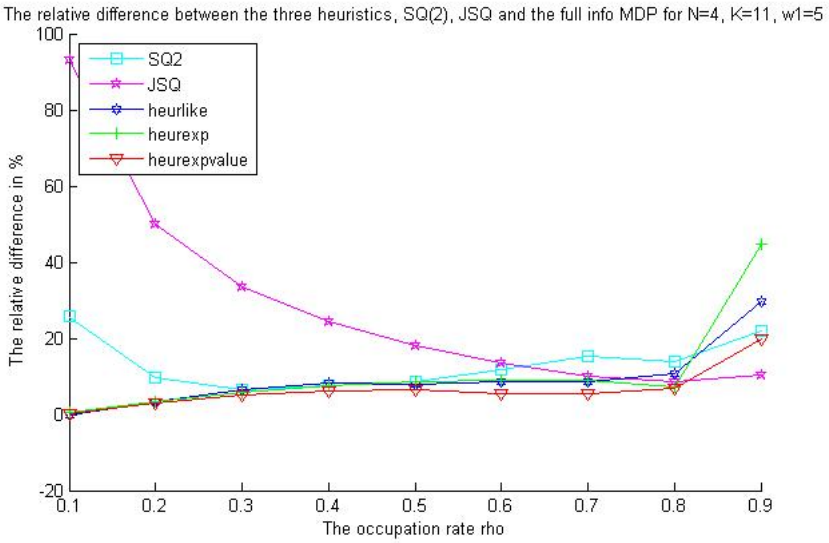


Figure 5.4: The performance of the three heuristics "heurlike", "heurexp" and "heurexpvalue" compared to the performance of SQ(2) and JSQ, measured in terms of the relative difference with respect to the MDP from Section 3, for $M = 4$ and $w_1 = 5$.

With regard to scalability however, these heuristics cannot be used directly, since they all use the MDP from Chapter 3, which is not suitable for high dimensional systems. Therefore, in the next section we will come up with a scalable heuristic on the basis of the ideas developed in this section.
Apart from the fact that we cannot use the MDP in Phase B, we should also take a closer look at the computation time in Phase A. Whereas "heurlike" requires the explicit computation of the M distributions of the subsystems, and "heurexpvalue" even requires the computation of distribution $\mathcal{Q}$[5], "heurexp" only requires to compute the expectation of the number of jobs per subsystem, which comes down to computing

$$\max\left(s_i - \mu\tau_i, 0\right) \tag{5.7}$$

per subsystem, since the expectation of a Poisson process with parameter μ in the interval $[0, \tau_i]$ is $\mu\tau_i$. So this grows linearly in M. Conclusively, despite not being the best performing of the three heuristics, the idea of "heurexp" is most suitable to work with in higher dimensional systems.

[5]To illustrate why this poses a problem, take the truncation level $K = 2$. For this relatively low truncation level, already 2^M computations are needed to compute $\mathcal{Q}$. That is, the number of computations grows exponentially in M and explodes for large M.

5.2.3 Approximation step 2: Getting rid of the MDP

From the previous section we have learnt that in Phase A it is most suitable, with respect to computation time, to compute x_e at every arrival. Moreover, we have observed that in Phase B we cannot use the MDP from Chapter 3 to compute a corresponding policy[6]. Hence, we have to come up with a solution to compute a reasonable policy differently, without using the MDP. To this end, observe that for $M = 1$ the supermarket model reduces to a regular M/M/1 queue, and the relative value function, denoted V^{MM1}, has a closed form expression (see [4]):

$$V^{MM1}(l) = \frac{l(l+1)}{2(\mu^{MM1} - \lambda^{MM1})}, \tag{5.8}$$

where λ^{MM1} is the arrival rate, μ^{MM1} the service rate and $l \in \mathbb{N}_0$ represents the number of jobs in the system. Recall that $p_j(x, d)$ is the probability that, in state x when applying SQ(d), an arriving job is sent to subsystem j. So the arrival rate for subsystem j is $\lambda p_j(x, d)$. We can use the above to approximate the relative value function V of the MDP of Chapter 3, for $M > 1$ as follows:

$$V(x) = \sum_{j=1}^{M} V^{MM1}(x, j), \tag{5.9}$$

where

$$V^{MM1}(x, j) := \frac{x_j(x_j + 1)}{2(\mu - \lambda p_j(x, d))}, \tag{5.10}$$

for fixed $d \leq M$ and $p_j(x, d)$ as defined in Definition 4. It is important to observe that V^{MM1} depends on the complete state x and not only on one of its components. That is, the arrival rate for subsystem j is dependent on the number of jobs in the other subsystems. We will come back to this below. Given (5.9), we can adjust (5.5) to

$$d^*(x) := \underset{d \in \mathcal{D}}{\operatorname{argmin}} \left\{ f_2(d) + \lambda \sum_{i=1}^{M-d+1} k(M, i, d) \sum_{j=1}^{M} V^{MM1}(\sigma(x + e_i), j) \right\}, \tag{5.11}$$

for $x \in \mathcal{X}$.
We want to emphasize that the interdependencies of the subsystems, when applying

[6]Because we will not use the MDP anymore, we do not have to truncate the state space either.

SQ(d) policies with $d > 1$, is one of things that make these algorithms very hard to analyze. These dependencies become increasingly significant for higher values of d, since more subsystems will be sampled. Also, the higher the occupation rate ρ, the better JSQ performs (in terms of the number of jobs waiting in the subsystems). In the approximation (5.9) of V, these interdependencies are merely accounted for through $p_i(x, d)$. Therefore, contrary to the "heurexp" algorithm in the previous section, we do not simply compute x_e and subsequently sample $d^*(\sigma(x_e))$ subsystems at every job arrival. Instead, after computing $d^* \equiv d^*(\sigma(x_e))$, we compute

$$d^{**} = \lfloor (1-\rho)d^* + \rho M \rfloor \,, \tag{5.12}$$

to account for the comments made above. That is, for higher occupation rates we expect d^* to be lower than it should be, since in its calculation we expect it to underestimate the interdependencies between the subsystems. Therefore, after computing d^* we apply the Bernouilli-like weighing (5.12) of d^* and M, to correct for this anticipated underestimation. Subsequently, d^{**} subsystems are sampled. The resulting heuristic comes down to the following:

> Calculate x^e at every arrival and sample $d^{**} = \lfloor (1-\rho)d^*(\sigma(x^e)) + \rho N \rfloor$ subsystems; we call this heuristic "*Dynamic SQ(d)*".

We could also have rounded $(1-\rho)d^* + \rho M$ up or to the nearest integer, instead of rounding it down. For our purposes, the conservative choice of rounding it down gave the best results.

Remark 7. *Complexity*

Before looking at the performance of our Dynamic SQ(d) heuristic in the section below, we would like to make a comment on the complexity of the algorithm. Since we want to apply the heuristic to high dimensional systems (i.e., large M), the complexity should not be too high. In order to analyze the complexity, it is also important to take into account what should be stored in memory. Apart from the $1 \times M$ vectors τ and s, it it sensible to compute $k(M, i, d)$ for all $i, d \in \{1, \ldots, M\}$ up front and store these values in memory[7] ($M \times M$ matrix). Then, the Dynamic SQ(d) has the following complexity. At every arrival:

[7] One might also consider computing all possible $p_i(x, d)$ up front. However, it does not scale to store these values since we do not truncate the state space here. Moreover, even if the state space was finite, then there would be an exponential number of possible states x. Hence we do not compute them up front but calculate them when needed.

+ It takes $\mathcal{O}(M)$ computations to calculate x_e;

+ It takes $\mathcal{O}(M \log M)$ computations to sort x_e;

+ It takes $\mathcal{O}(M^4 \log M)$ computations to calculate $d^*(x)$, for $x \in \mathcal{X}$:

  $\mathcal{O}(M \log M)$: Calculating $\sigma(x + e_i)$ for fixed i and $x \in \mathcal{X}$,

  $+\mathcal{O}(M)$: Calculating $p_j(\sigma(x + e_i), d)$ for fixed d, i, j and $x \in \mathcal{X}$,

  $*\mathcal{O}(M)$: We sum over $j \in \{1, \ldots, M\}$,

  $*\mathcal{O}(M)$: We sum over $i \in \{1, \ldots, M - d + 1\}$,

  $*\mathcal{O}(M)$: We do this for every $d \in \{1, \ldots, M\}$.

+ It takes $\mathcal{O}(1)$ to calculate d^{**};

+ It takes $\mathcal{O}(M)$ to apply SQ(d^{**});

+ It takes $\mathcal{O}(M)$ to update τ and s.

Conclusively, the Dynamic SQ(d) heuristic is an algorithm with polynomial complexity $\mathcal{O}(M^4 \log M)$, where we have assumed that the sorting algorithm $\sigma(.)$ has complexity $\mathcal{O}(M \log M)$. Possibly, in this specific case we can bring down the complexity of the sorting algorithm to $\mathcal{O}(M)$, yielding an overall complexity of $\mathcal{O}(M^4)$.

Although the Dynamic SQ(d) heuristic has a polynomial complexity, its order is quite high. Due to this, simulations for very large M will still be rather computationally intensive. For this reason, we have restricted ourselves to $M = 30$ in the simulations we have done. Reducing the complexity would allow for simulations for much higher M.

Remark 8. *Accuracy*

The information state we compute in Phase A at every arrival, is in fact a probability distribution that represents the likelihood that the process is in a particular state[8]. In general, the accuracy of these information states tends to deteriorate as the process progresses due to accumulated errors. In our case however, the accuracy of the information state tends to improve as more subsystems are sampled. In fact, whenever a subsystem is sampled, all probability mass is concentrated on the true number of jobs in that subsystem. Thus, the higher the value of d in SQ(d), the more

[8]For Dynamic SQ(d) we do not compute the complete distribution, but just its expectation.

subsystems are sampled and consequently the higher the accuracy of the information state, which in turn should improve the performance of the heuristic. Obviously, on the other hand increasing d has the consequence of increasing communication costs.

5.2.4 Results

In this section we will compare Dynamic SQ(d) with the algorithms SQ(2) and JSQ[9] for a system with $M = 30$ subsystems. Furthermore, we will distinguish between three different scenarios, representing different types of networks. Concretely, we will look at $w_1 = 5, w_1 = 40$ and $w_1 = 80$. For $w_1 = 5$ communication costs are relatively high, whereas for $w_1 = 80$ they are relatively low.
In Figure 5.5 we give the results of this comparison. It can be concluded that the Dynamic SQ(d) heuristic outperforms standard algorithms SQ(2) and JSQ for the scenarios with relatively low communication costs (i.e., $w_1 = 40, 80$), whereas for scenarios where the communication costs are relatively high ($w_1 = 5$), SQ(2) performs considerably better than our heuristic. In fact, our heuristic is a hybrid version of SQ(2) and JSQ, since it chooses all its values between $d = 2$ and $d = M$[10].

Conclusively, from Figure 5.5 it can be concluded that choosing d dynamically (according to the Dynamic SQ(d) algorithm) instead of statically, gives good results in scenarios where communication costs are relatively low[11]. Also, remark that as the occupation rate increases, the Dynamic SQ(d) algorithm moves to a JSQ policy more and more.

As can be seen in Figure 5.5, Dynamic SQ(d) does not perform well for all scenarios. That is, it does not perform very well in the scenario where communication cost are relatively high, and we think it is because of the way d is computed. Indeed, after calculating d^*, we compute d^{**} according to formula (5.12), which we give here again, for convenience:

$$d^{**} = \lfloor (1 - \rho)d^* + \rho M \rfloor .$$

[9] As we have seen before, RR and SQ(1) perform generally very badly. This is also the case for the sets of parameters considered in this section, hence we have left them out of the comparison.

[10] $d = 1$ will hardly be chosen by Dynamic SQ(d), due to the way d^{**} is computed. This will be illustrated later on.

[11] This conclusion is obviously dependent on the cost function we have chosen. However, we expect that also for more realistic cost functions this conclusion remains valid, but this would have to be investigated in future research.

Thus, if M is large, it dominates the computation of d^{**} and small values for d^{**} are not possible, even for low occupation rates. For instance, assume $M = 30$, $\rho = 0.1$ and $d^* = 1$. Then $d^{**} = \lfloor 0.9 + 3 \rfloor = 3$ is the minimum value for d^{**}[12], whereas choosing $d = 1$ or $d = 2$ may be sufficient with respect to minimizing the maximum queue length, for low occupation rates ρ. In the scenario where communication costs are relatively low, choosing a higher d does not cost as much in relative terms and balancing the load well is of much higher importance. Consequently, Dynamic SQ(d) performs better in these cases. If M is even larger, the just described effect will become ever more prominent. Thus, although we had good arguments to support computing d^{**} as in (5.12), in some cases there might be a need to adapt this formula. An extreme case would be to omit computing d^{**} and apply Dynamic SQ(d) with d^* anyway, despite the arguments given. Some initial tests (see Figure 5.6) show that in the scenarios given above, this adapted version of Dynamic SQ(d) outperforms Dynamic SQ(d) for quite some occupation rates. Ideally, there should be a trade-off between the two extremes and this could be topic of future research. Since this observation should be experimented with extensively, in the rest of this thesis we continue with the Dynamic SQ(d) algorithm as given.

It is not the first time that a dynamic SQ(d) policy has been proposed. In [6] the so-called "d-adaptive algorithm" is suggested. In order to explain this algorithm, let $f(k)$ be a positive integer-valued, non-decreasing function of k. For every arrival, do the following:

1. Choose a subsystem at random.

2. Suppose this subsystem has k jobs. Sample $f(k)$ additional subsystems so that the total number of samples equals $f(k) + 1$.

3. Send the arrival to the shortest of these $f(k) + 1$ queues, breaking ties at random.

In [6] $f(k) = k$ and $f(k) = k^2$ are suggested, for which the performance is shown in Figure 5.7. For the first scenario, in contrast to Dynamic SQ(d), for low occupation rates the d-adaptive algorithm does not suffer from having to choose too high d's, and it is the best performing algorithm considered. However, for the other two scenarios, the d-adaptive algorithms (just like SQ(2) and JSQ) are outperformed by Dynamic SQ(d). In these scenarios, we expect that this improved performance is due to the

[12]This specific example also shows why we have chosen to round down instead of rounding up or rounding to the nearest integer in the computation of d^{**}.

fact that for higher occupation rates it is more important to choose a good value of d. Since Dynamic SQ(d), contrary to the other algorithms considered, takes into account information from the past, it is better capable of choosing a good d than the other algorithms, that do not take the past into account.
Two remarks are in place here. Firstly, the d-adaptive algorithm seems to have the same performance for both $f(k) = k$ and $f(k) = k^2$ for low occupation rates. Obviously, for low occupation rates it is very likely that the subsystem sampled initially has $k = 0$ or $k = 1$ jobs, and both algorithms perform the same since $k = k^2$ in these cases. Secondly, the d-adaptive algorithm in fact uses two rounds of communication, which is something we have not taken into account in the cost function we have used here. Indeed, first sampling one subsystem and subsequently $f(k)$ others will most likely cost more time than sampling $f(k) + 1$ straight away. The current cost function is not capable of taking this into account.

5.2.5 Some ideas for improving Dynamic SQ(d)

We want to finish this section by proposing two ideas for improvement of the Dynamic SQ(d) algorithm. The first has to do with reducing its complexity, and the second with improving its performance.

In Remark 7 we discussed the complexity of Dynamic SQ(d). We saw that one of the $\mathcal{O}(M)$ terms when computing d^* in (5.11), was due to minimizing over $d \in \{1, \ldots, M\}$. Our idea is that if we could show that d^* only takes on certain values for certain parameters, then we could minimize over this set instead. To illustrate this idea, in Table 5.2 we have kept track of the value of d^* computed for the Dynamic SQ(d) algorithm for $M = 30$, $w_1 = 40$ and $\rho = 0.1, 0.5$ and 0.9. The corresponding values of d^{**} are also given. We see that for $\rho = 0.1$ it is the case that d^* takes on only the values $1, 2$ and 3. For $\rho = 0.5$ and $\rho = 0.9$, however, there is not such a clear interval of values of d^* that are used. What might help us though, is that when calculating d^{**}, different values of d^* may be mapped to one value of d^{**}. How we could exactly use these ideas might be investigated in future research.

	$\rho = 0.1$	$\rho = 0.5$	$\rho = 0.9$
Values of d^*	1-3	2-15, 17,20,23,28,30	1-30
Values of d^{**}	3-5	16-23, 25, 26, 29, 30	27-30

Table 5.2: All values of d^* that are used by Dynamic SQ(d) for $M = 30$, $w_1 = 40$ and $\rho = 0.1, 0.5$ and 0.9, and the corresponding d^{**}.

The second idea has to do with the way we use the information we have. Up till now, we have restricted ourselves to allowing an arriving job to go only to one of the d sampled subsystems. Since we calculate the expected number of jobs at every arrival, we could also use this information to send a job to one of the subsystems outside the sample of SQ(d). We call the algorithm that follows this idea SQ+(d). In Figure 5.8 we have experimented a bit with this for SQ(2) and SQ+(2), but more research is needed here. We can see that SQ+(2) performs only better than SQ(2) for high occupation rates. One idea to improve its performance for lower occupation rates, is to also take the variance into account, since the higher the variance, the higher the uncertainty about the expectation. If two subsystems have the same expected number of jobs, but the variance for one of them is higher, it may be sensible to send the job to the one with the lowest variance. These ideas may also be investigated further in future research.

5.3 Different service rates

Contrary to the situation we have discussed so far, in this section we will consider our Dynamic SQ(d) heuristic for the supermarket model with unequal service rates, which represents reality better. That is, instead of assuming $\mu_1 = \mu_2 = \cdots = \mu_M \equiv \mu$ for some $\mu > 0$, in this section we allow for $\mu_i \neq \mu_j$, $i \neq j$. Consequently, we should no longer take the maximum queue length as part of our costs, since we have different service rates. Instead, we should look at the remaining work per subsystem, which is defined as the number of jobs in the subsystem, divided by the service rate, i.e., x_i/μ_i[13]. Now we have two options to continue:

1. We could assume that the service rates are known to the decision maker (i.e., the dispatcher in our case). This corresponds to what we have assumed before in this thesis, since the service rates are used in the calculation of V^{MM1} (see expression (5.10)), which is part of the computations for the Dynamic SQ(d) algorithm. In order for Dynamic SQ(d) to be used in this setting, we have to slightly adapt it. Calculating x_e can be done similarly to (5.7). That is, given vectors τ and s, we can compute

$$\max\left(s_i - \mu_i \tau_i, 0\right) \tag{5.13}$$

   for every subsystem, which yields x_e. Lastly, only the computation of $V^{MM1}(x, j)$

[13]Recall that due to memoryless property of the exponentional service rates, it does not matter how long the job is service has already been processed.

should be changed slightly, compared to (5.10). Here,

$$V^{MM1}(x, j) := \frac{x_j(x_j + 1)}{2(\mu_j - \lambda p_j(x, d))}, \tag{5.14}$$

with $p_j(x, d)$ as defined in (3.11).

2. We could assume that the service rates are not known. In this situation we could estimate the service rates while controlling the system, by keeping a distribution for these service rates per subsystem. A commonly used distribution for this purpose is the Gamma distribution, because it is closed under a Bayesian update in this case. These distributions could then be used to calculate the expected values of $V^{MM1}(x_e, j)$, for each j. That is, given x_e, we could compute $\mathbb{E}_{\mu_j}\left[V^{MM1}(x_e, j)\right]$ for each $j \in \{1, \ldots, M\}$, where $\mathbb{E}_{\mu_j}$ denotes the expectation with respect to the distribution for service rate μ_j.

We will continue here with the first option, although the second option is also very interesting as it might reflect reality better. This involves considerably more research, however, and might be a topic for future research. A good reference is [12].

We have to emphasize that the SQ(d)-related algorithms send a job to a particular subsystem on the basis of the *number of jobs* per subystem, rather than on the *remaining work* per subsystem. The algorithm that bases its decision on the remaining work per subsystem is called "Least Loaded of d queues" (denoted LL(d)). In the case where all service rates are equal, SQ(d) and LL(d) are equivalent. Consequently, making the step to modelling the supermarket model with different service rates, also requires making a choice between applying SQ(d) or LL(d). LL(d) should outperform SQ(d), because it uses more information on the real workload per subsystem. Figure 5.9 supports this claim by showing it for $d = 2$ as well as for Dynamic SQ(d) and Dynamic LL(d), where Dynamic LL(d) is the straightforward generalization of Dynamic SQ(d). That is, instead of computing $p_i(x, d)$, one needs to compute $p_i(x^\mu, d)$, where x^μ represents the remaining workload per subsystem, i.e., $x_i^\mu = x_i/\mu_i$. Then, after computing d^{**}, instead of applying SQ(d^{**}) it applies LL(d^{**}).
Given this observation, it is advisable to continue with Dynamic LL(d) instead of Dynamic SQ(d). For completeness, and because it might be relevant for the case where the service rates are not known, we have considered both in the section on results below.

5.3.1 Results

In this section we will, similarly to Section 5.2.4, compare the Dynamic SQ(d) algorithm with SQ(d) and JSQ. However, here the servers are divided in three groups of ten servers. The service rate for the first group of servers is 0.5, that of the second group 1, and for the last group 1.5. Furthermore, we will compare the Dynamic LL(d) algorithm with LL(d) and LL(30).
In Figure 5.10 we see that for different service rates, Dynamic SQ(d) performs better compared to SQ(2) and JSQ, than for the case with equal service rates, in particular for $w_1 = 5$. This is not surprising, since Dynamic SQ(d) exploits the knowledge about the different service rates when determining d^{**} (see Equations (5.13) and (5.14)), whereas SQ(d) policies do not take into account any knowledge about the service rates. Similar to the case with equal service rates, one can see that for $w_1 = 5$ and low occupation rates, also here Dynamic SQ(d) suffers from choosing to high values of d due to the way d^{**} is computed.
For Dynamic LL(d), this observation is less beneficial since LL(d) policies do take the service rates into account when sampling the subsystems. It is only in the determination of d^{**} that Dynamic LL(d) might outwit static LL(d) policies. This situation is similar to the situation with equal service rates, as can also be seen in the performance comparison in Figure 5.11.

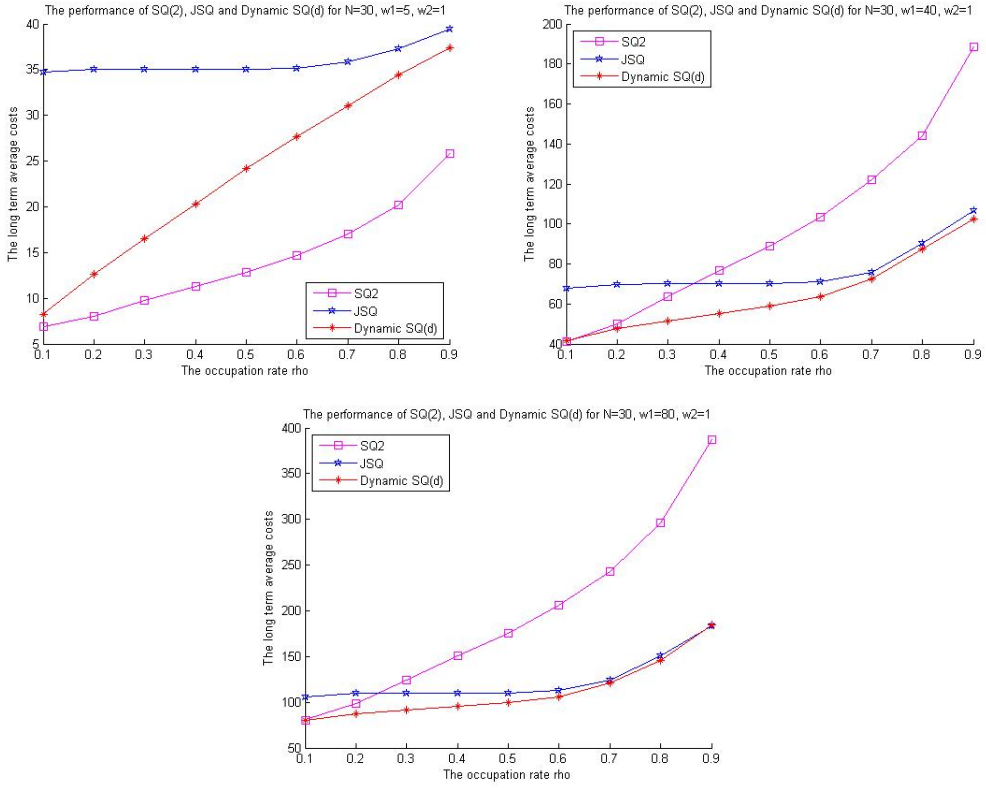


Figure 5.5: Performance of the Dynamic SQ(d) heuristic compared with SQ(2) and JSQ, for $M = 30$ and $w_1 = 5$ (top left), $w_1 = 40$ (top right) and $w_1 = 80$ (bottom). In the first scenario (corresponding to relatively high communication costs), Dynamic SQ(d) does not perform very well compared to the other two, but in the other two scenarios (corresponding to relatively low communication costs), it outperforms SQ(2) and JSQ.

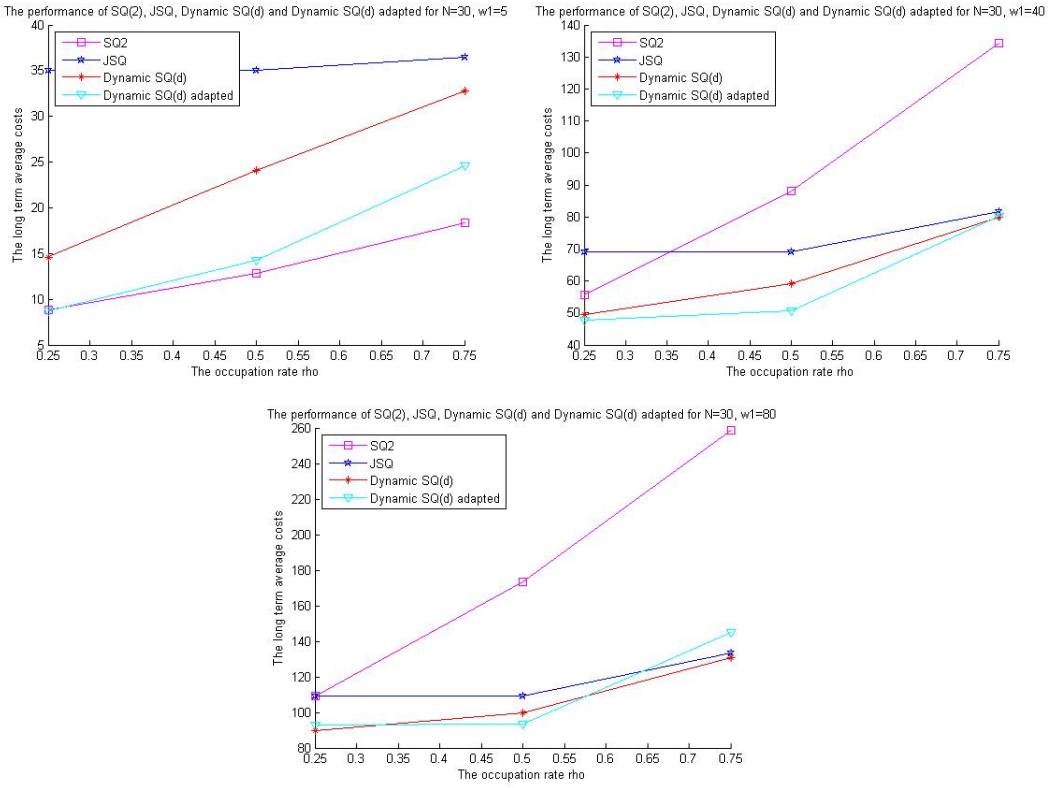


Figure 5.6: Performance of the Dynamic SQ(d) heuristic compared to its adapted version, for $M = 30$ and $w_1 = 5$ (top left), $w_1 = 40$ (top right) and $w_1 = 80$ (bottom). In the adapted version, instead of using d^{**} to perform SQ(d) with, it uses d^*. For completeness, SQ(2) and JSQ are also given.

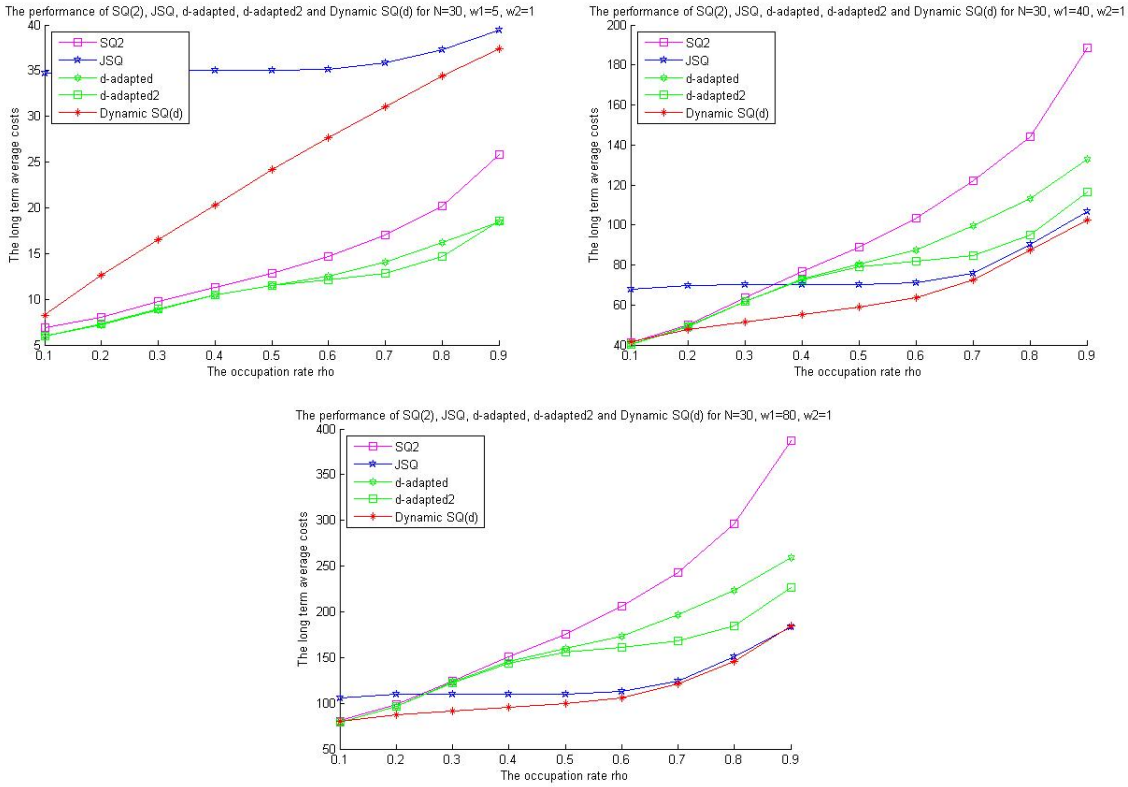


Figure 5.7: Performance of the d-adaptive algorithm with $f(k) = k$ and $f(k) = k^2$ compared with the Dynamic SQ(d) heuristic and SQ(2) and JSQ, for $M = 30$ and $w_1 = 5$ (top left), $w_1 = 40$ (top right) and $w_1 = 80$ (bottom).

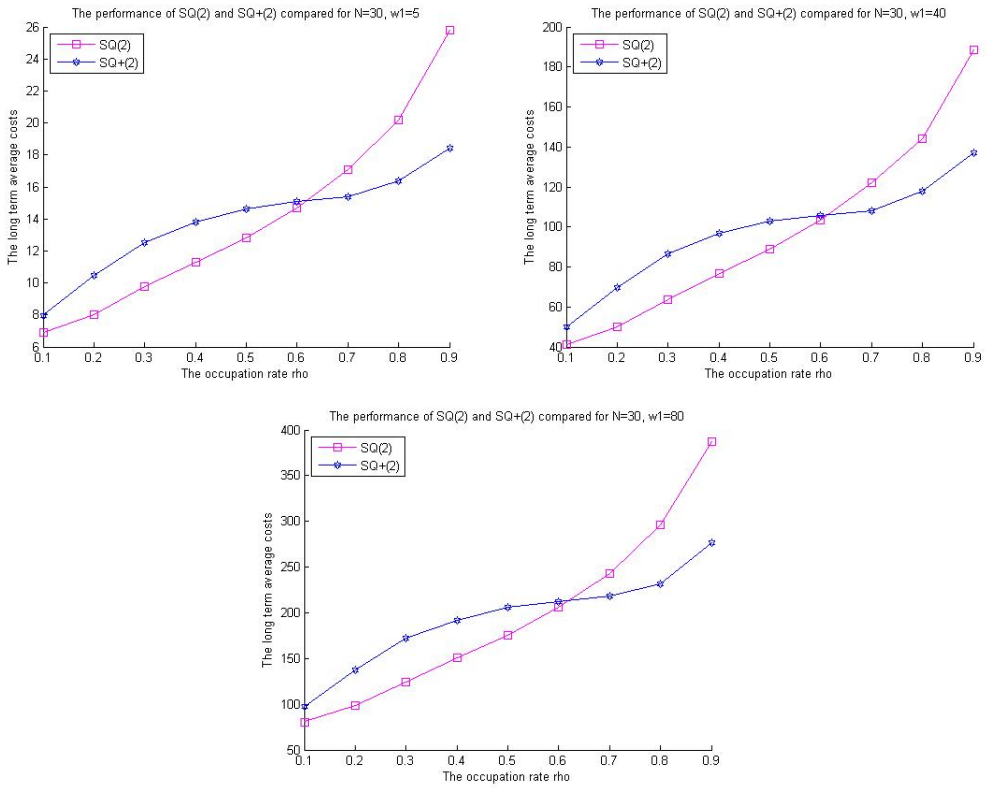


Figure 5.8: Comparison of the performance of SQ(2) and SQ+(2), for $M = 30$ and $w_1 = 5$ (top left), $w_1 = 40$ (top right) and $w_1 = 80$ (bottom).

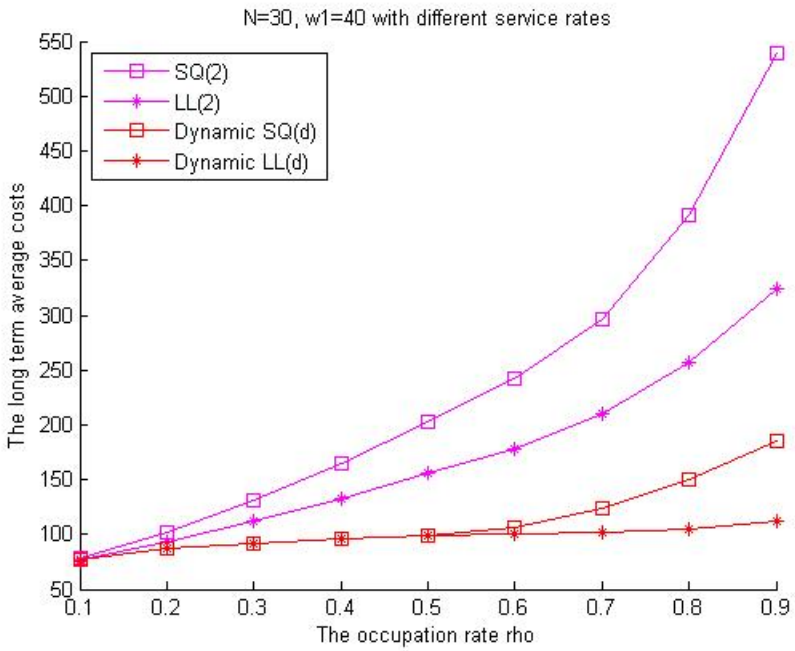


Figure 5.9: In this picture the observation that LL(d) should outperform SQ(d) is illustrated for $d = 2$, as well as for Dynamic SQ(d) and Dynamic LL(d).

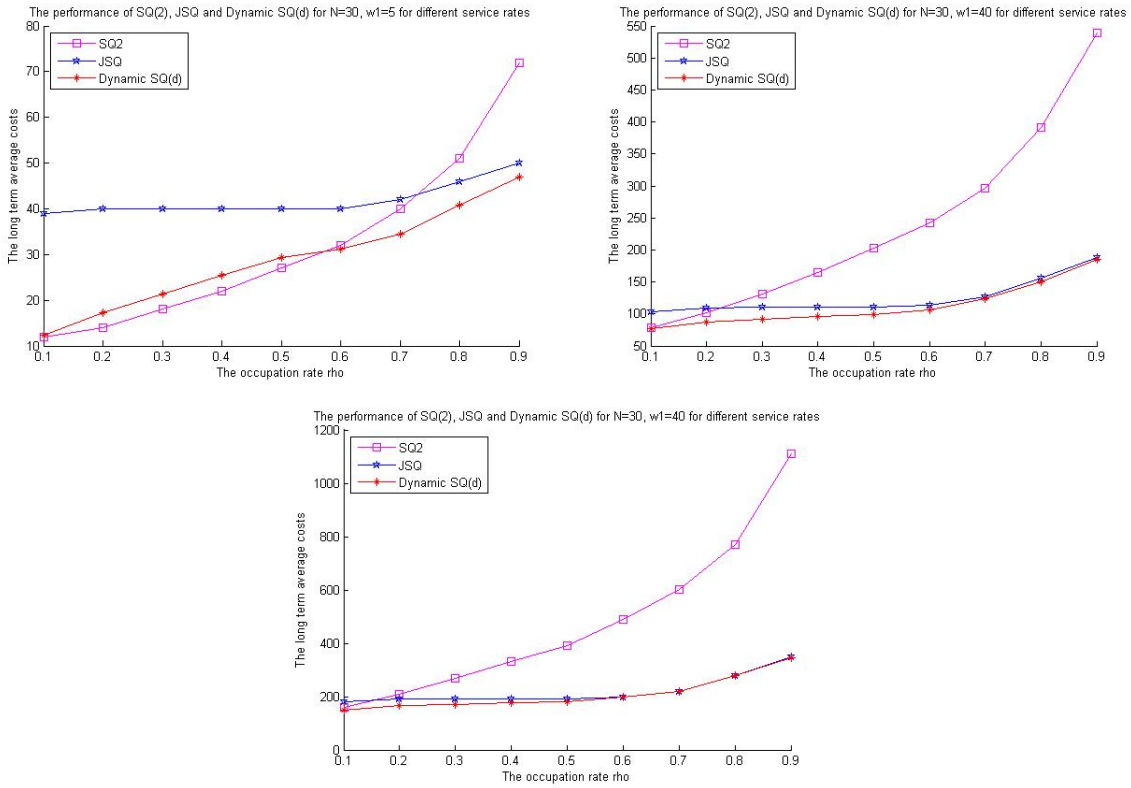


Figure 5.10: Performance of the Dynamic SQ(d) algorithm compared to SQ(2) and JSQ, for $M = 30$ and $w_1 = 5$ (top), $w_1 = 40$ (middle) and $w_1 = 80$ (bottom), in the case with different service rates. The servers are divided in three groups of ten servers. The service rate for the first group of servers is 0.5, that of the second group 1, and for the last group 1.5.

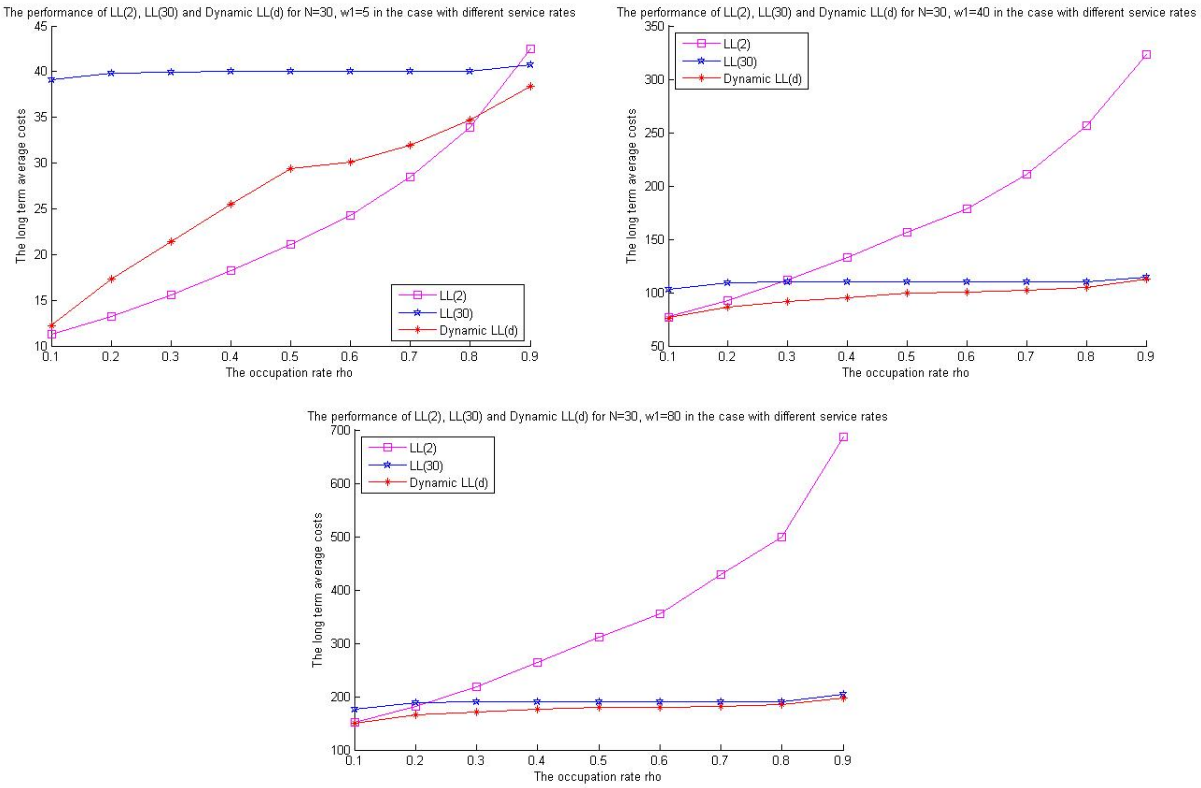


Figure 5.11: Performance of the Dynamic LL(d) algorithm compared to LL(2) and LL(30), for $M = 30$ and $w_1 = 5$ (top), $w_1 = 40$ (middle) and $w_1 = 80$ (bottom), in the case with different service rates. The servers are divided in three groups of ten servers. The service rate for the first group of servers is 0.5, that of the second group 1, and for the last group 1.5.

Chapter 6

Conclusions, discussion and future research

In this chapter we provide some conclusions and ideas for future research. Most of these observations have already been mentioned in the main text, but we summarize and extend them here.

6.1 Conclusions and discussion

To the best of our knowledge, this is the first time that the theory of Markov decision processes has been applied to the topic of randomized load balancing in the way we do in this thesis. We have shown that Markov decision theory is suitable as modelling tool in this area and that it can add to the existing literature because it refrains from asymptotics and allows for more flexibility in the approach of randomized load balancing. With the latter we mean in our particular case that Markov decision theory allowed us to let go of the fixed SQ(d) policies that are mostly studied in literature. Using Markov decision theory, we could investigate the influence of communication costs explicitly, something which has not been done before, as far as we know. By making the costs a weighted sum of these communication costs and the costs due to the queue lengths, we were able to show that choosing d dynamically outperforms choosing a fixed d, at least when it is done according to the optimal policy of the corresponding MDP. Since this MDP uses full information and is not scalable to high dimensions, we could not use this optimal policy directly. However, as we saw that learning about the system and choosing d dynamically could have a positive effect on the performance, we decided to make a heuristic that exploits this idea. To deal with the partial information issue, we developed a framework to estimate the state

of the system at every arrival. Subsequently, we approximated the optimal policy of the MDP by using that the relative value function of an M/M/1 queue has a closed form expression. The resulting heuristic, coined Dynamic SQ(d), outperforms traditional algorithms such as Round Robin, Random, SQ(2) and Join the Shortest Queue (JSQ) for the scenarios where communication costs are relatively low. Although its complexity is polynomial, its order is quite high, namely $\mathcal{O}(M^4 \log M)$, with M the number of servers in the system. In contrast, SQ(d) algorithms have a complexity of $\mathcal{O}(M)$. This relatively high complexity of Dynamic SQ(d) is a direct consequence of our idea to use the information from sampling to determinine a good d. In practice this computation time really adds to the total response time. This is something which the cost function we used is not able to take into account and which we have thus not considered in this thesis. Consequently, it is very important to try to reduce this complexity.

In the above, we have throughout assumed that we had full information about the parameters of the supermarket model. That is, we assumed that λ and μ are known to the dispatcher. When extending our models to the case with different service rates, the question of whether the service rates are known to the dispatcher, becomes crucial since the number of jobs in a subsystem is generally not equivalent to the remaining workload in the subsystem anymore. In case the service rates are known, we see that for the scenario where communication costs are relatively high, Dynamic SQ(d) performs much better than in the case with equal service rates, relative to SQ(2) and JSQ. This is due to the fact that Dynamic SQ(d) takes the information about the service rates into account whereas SQ(d) does not. When comparing Dynamic LL(d) with LL(d) policies, this advantage vanishes to some extend, and the Dynamic LL(d) performs comparable to Dynamic SQ(d) for the case with equal service rates.
In case the service rates are not known, which is probably the most realistic case, we could try to estimate the service rates and it becomes very interesting to see whether our heuristic can benefit enough from the information it uses to outperfom the traditional algorithms in this case too. We have postponed investigating this to future research.

With regard to the cost function, due to its particular structure it will, for fixed occupation rate, behave differently for different system sizes. Recall that the cost function is of the form $w_1 x_M + w_2 d$. The first part, $w_1 x_M$, is approximately constant in M, since for fixed occupation rate, intuitively the maximum queue length x_M will probably not change very much for different M. Consequently, $w_1 x_M$ will not change

very much when increasing M from say $M = 4$ to $M = 30$. The second part of the costs, $w_2 d$, may change much more, since $d \in \{1, \ldots, M\}$ and d may thus take on value M. $w_2 d$ scales linearly in d, whereas in practice one might expect $f_2(d)$ to be a concave function in d. That is, since the dispatcher samples subsystems in parallel, one expects the incremental cost of sampling one subsystem more, small compared to the cost of sampling only 1 subystem. Studying more realistic cost structures can be done in future research. In these more sophisticated cost functions, also the complexity of calculating a good d for Dynamic SQ(d), should be included, as mentioned above.
In order to make the computation time of the optimal policy of the MDP feasible for higher dimensions, it is worthwile studying structural properties of its relative value function V. Showing non-decreasingness and convexity of V can be used to compute the optimal policy of the MDP with lower complexity. We have managed to prove non-decreasingness and expect V to be convex as well, but we have not yet succeeded proving this using standard techniques. New techniques may have to be developed for this. Although the MDP uses full information and may still not be scalable up to the dimensions we are interested in, improving computation speed might be of great use in reducing the complexity of our heuristics as well.

The main conclusion we can draw from this thesis is that although in practice often SQ(2) is chosen to perform the load balancing in large systems, in certain scenarios it pays off to also consider communication costs and choose d dynamically, using the information we receive from sampling. This has led to a promising heuristic that is scalable up to $M = 30$ subsystems. We believe this heuristic can be the basis for a well performing heuristic that is scalable to even larger systems ($M > 100$).

6.2 Future research

During the course of writing this thesis I have come across many things that are worthwile studying in future research. I will list them below.

- In Chapter 5 we already considered the supermarket model with different exponential service rates. It would be very interesting to extend this further to general service distributions and arrival processes. Also, the FIFO service discipline might swopped for another, perhaps more realistic, service discipline such as Processor Sharing (PS).
  With regard to generalizing the service distributions, it would be natural to first look at phase-type distributions, since these are closed in the set of all non-

negative distributions and fairly easily modelled using Markov decision theory. However, extending to these phase-type distributions might come at the cost of adding additional dimensions to the state space and therefore increasing the complexity of the MDP.

- Instead of minimizing the maximum queue length as part of our cost objective, it might be good to look at other cost objectives as well. We can think of e.g., minimizing the *average* queue length, which also leads to minimizing response times but may result in a slightly different way of doing so and thus giving new insights.

- Instead of the linear cost function we have considered, taking a more sophisticated cost structure (i.e., non-linear) might be more realistic and may yield new insights. It would be interesting to see whether the conclusions we have drawn still hold for different cost functions. In particular if we include the computation time for computing a good d in Dynamic SQ(d) in this cost function.

- It remains an open question whether the relative value function of the MDP we formulated in Chapter 3 in indeed convex. New techniques may be developed to prove this.

- Another way of dealing with the curse of dimensionality of MDPs, is by applying approximate dynamic programming. This consists of combining statistical learning (e.g., regression or neural networks), simulation and combinatorial optimization. The idea is that we approximate the relative value function V by for instance a polynomial function. A good reference in this area is [18].

- In the formulation of the Markov reward and Markov decision models in Chapter 3, we assumed a sorted state space. As a consequence, amongst other things, the transition probabilities had a fairly simple form. If we do not sort the state space, the model will change and perhaps become more difficult to formulate, but this may lead to new insights as well.

- We have opted in this thesis to solve the Poisson equation (3.8) and optimality equation (3.15) using value iteration, as explained in Section 2.2.1. We could also have used linear programming to formulate and solve the problem, but this is generally hard if not impossible as the system to solve has $|X|$ independent equations, where $|X| = \infty$ if we do not truncate the state space. Nevertheless, it might be worthwhile to look into this.

- As mentioned in Section 5.3, if the service rates are unknown we could use techniques to estimate these service rates by keeping a distribution for every service rate. A good reference in this area is [12].

- In contrast with standard literature, we have included communication costs in the cost objective. Alternatively, instead of one cost objective that weighs the communication costs and the queue size costs, we could reformulate the model such that it has two objective functions. Usually, minimizing one objective under constraints on the other objective is the way to go. Concretely, we would be minimizing the maximum queue length under constraints on the communication costs. This could lead to an algorithm that samples all servers and waits for a fixed period (such that the communication constraint is not violated) and applies SQ(d), where d in this case would be the number of servers that have given feedback within the fixed period of time. A good reference to these constrained Markov decision chains is [2].

- More investigation into the SQ+(d) algorithm that we proposed in Section 5.2.5, might result in algorithms that combine the simplicity of SQ(d) with the estimation techniques used in Dynamic SQ(d).

- In Section 5.2.5 we also proposed keeping track of the values of d^* computed by Dynamic SQ(d). If a pattern can be distinguished in this respect, this might lead to a reduction in complexity of Dynamic SQ(d) since the minimization can be done over a smaller set than $\mathcal{D} = \{1, \ldots, M\}$.

- In Section 5.2.4 we saw that the computation of d^{**} in case M is large, resulted in high values of d^{**}, which particularly decreased performance for low occupation rates. Future research might be devoted to finding a formula for d^{**} so that Dynamic SQ(d) also performs well for this range of occupation rates, without losing too much performance in the scenarios where it performs well already.

- Finally, we should test the Dynamic SQ(d) algorithm with real data.

Affiliation

Jöbke Janssen
Ina Boudier Bakkerlaan 69 IV
3582 VV Utrecht, The Netherlands
Telephone: +31653548705
E-mail: jobkejanssen@gmail.com
Student number: 3128474

References

[1] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions, with Formulas, Graphs, and Mathematical Tables*, volume 55. 1972.

[2] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall, 1999.

[3] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. *Siam J. Comput*, 29(1):180–200, 1999.

[4] S. Bhulai. *Markov Decision Processes: the control of high-dimensional systems*. PhD thesis, Vrije Universiteit Amsterdam, 2002.

[5] S. Bhulai and G. Koole. *Stochastic Optimization*. VU University Amsterdam, Department of Mathematics, 2010.

[6] M. Bramson and B. Lu, Y. Prabhakar. Randomized load balancing with general service time distributions. *Sigmetrics*, 2010.

[7] A. Ganesh, S. Lilienthal, D. Manjunath, A. Proutiere, and F. Simatos. Load balancing via random local search in closed and open systems.

[8] R. Jain. *Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling*. Wiley Computer Publishing, John Wiley and Sons, Inc., 1991.

[9] R. Karp, M. Luby, and F. Meyer auf der Heide. Efficient pram simulation on a distributed memory machine. *Proc. 24th Annual ACM Symposium on the Theory of Computing*, pages 318–326, 1992.

[10] G. Koole. A transformation method for stochastic control problems with partial observations. *Systems and Control Letters*, 35:301–308, 1998.

[11] G. Koole. Monotonicity in markov reward and decision chains: Theory and applications. *Foundations and Trends in Stochastic Systems*, (1):1–76, 2006.

[12] P. Kumar. A survey of some results in stochastic adaptive control. *SIAM Journal of Control and Optimization*, 23:329–380, 1985.

[13] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. Larus, and A. Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation*, 68:1056–1071, 2011.

[14] M. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, Harvard University, 1991.

[15] M. Mitzenmacher. On the analysis of randomized load balancing schemes. *Theory of Computing Systems*, 31:361–386, 1999.

[16] M. Mitzenmacher. How useful is old information? *IEEE Transactions on Parallel and Distributed Systems*, 11(1), 2000.

[17] M. Mitzenmacher, D. Shah, and D. Prabhakar. Load balancing with memory. *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 799–808, 2002.

[18] W. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (2nd Edition)*. Wiley Series in Probability and Statistics, 2011.

[19] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Interscience, John Wiley and Sons, Inc., 1994.

[20] L. Quan-Lin, C. John, and Y. Wang. A matrix-analytic solution for randomized load balancing models with phase-type service times. *Information Systems*, 2011. Submitted.

[21] E. Schurman and J. Brutlag. The user and business impact of server delays, additional bytes and http chunking in web search. *O'Reilly Velocity Web*, 2009.

[22] D. Shah and B. Prabhakar. The use of memory in randomized load balancing. *Proceedings of the ISIT*, page 125, 2002.

[23] B. Vocking. How asymmetry helps load balancing. *IEEE Symp. Found. Comp. Sci*, pages 131–140, 1999.

[24] N. Vvedenskaya, R. Dobrushin, and F. Karpelevich. Queueing system with selection of the shortest of two queues: an asymptotic approach. *Problems of Information Transmission*, 32:15–27, 1996.

[25] R. Yang. *Adaptive Resource Allocation in High-Performance Distributed Multimedia Computing*. PhD thesis, VU University Amsterdam, 2011.

Appendix

A.1 Convexity

As has been mentioned in Chapter 4, we expect the relative value function V, of the dynamic SQ(d) algorithm, to be convex. However, our attempts using standard techniques to prove this property, have not yet been successful. It can be concluded that either V is not convex anyway, or new techniques must be developed to prove convexity in this case. In this appendix we will show the different standard ways that we have used to try to prove convexity.

Convexity of a function $f : \mathbb{N}_0^M \to \mathbb{R}$ is generally defined as

$$f(x + e_j) - 2f(x) + f(x - e_j) \geq 0,$$

for all $1 \leq j \leq M$ and all $x \in \mathbb{N}_0^M$ such that $x_j \geq 1$ (see also [11]). In our case, convexity of the relative value function V comes down to

$$V(\sigma(x + e_j)) - 2V(x) + V(\sigma(x - e_j)) \geq 0,$$

for all $1 \leq j \leq M$ and all $x \in \mathcal{X}$ such that $x_j \geq 1$.

In what follows, we need the following lemma.

Lemma A.1.1. *Fix some $n \geq 0$. Given that*

$$V_n(\sigma(x + e_j)) - 2V_n(x) + V_n(\sigma(x - e_j)) \geq 0, \tag{A.1}$$

for all $x \in \mathcal{X}$ such that $x_j \geq 1$, with $j = 1, \ldots, M$, also

$$V_n(\sigma(\sigma(x + e_j) - e_i)) - 2V_n(\sigma(x - e_i)) + V_n(\sigma(\sigma(x - e_j) - e_i)) \geq 0 \tag{A.2}$$

and

$$V_n(\sigma(\sigma(x + e_j) + e_i)) - 2V_n(\sigma(x + e_i)) + V_n(\sigma(\sigma(x - e_j) + e_i)) \geq 0, \tag{A.3}$$

for $i \in \{1, \ldots, M\}, i \neq \underline{j}, j, \overline{j}$, with $\underline{j}$ and $\overline{j}$ as defined in Definition 5.

Proof. We will only prove (A.2) as the proof of (A.3) is similar. Fix j and $i \neq \underline{j}, j, \overline{j}$. Basically, we have to show that the left hand side of (A.2) is equal to

$$V_n(\sigma(x^* + e_h)) - 2V_n(x^*) + V_n(\sigma(x^* - e_h), \tag{A.4}$$

for some $x^* \in \mathcal{X}$ and $h \in \{1, \ldots, M\}$, because then we can use (A.1).
We will assume that indices l exist with $x_l = x_j \pm 1$. In particular, we will assume that $1 < j < M$. If this is not the case, the proof will only become easier.
Similarly to Lemma 4.1.2, we define $x^* := \sigma(x - e_i)$. Recall that by Lemma 4.1.1 we know that there exists an $\underline{i}$ with $1 \leq \underline{i} \leq i$ such that $\sigma(x - e_i) = x - e_{\underline{i}}$ (i.e., we take $x^* := \underline{x}^i$). From Lemma 4.1.2 we have that for $i > \overline{j}$ or $\underline{j} \leq i < \overline{j}$, $\underline{j} < j$ or $i < \underline{j}$ we can take $h = j$ to get $\sigma(\sigma(x + e_j) - e_i) = \sigma(x^* + e_h)$. For the case $\underline{j} \leq i < \overline{j}$, $\underline{j} < j$ taking $h = j+1$ suffices. What remains to show is that $\sigma(\sigma(x - e_j) - e_i) = \sigma(x^* - e_h)$ with h as above for the corresponding $i \neq \underline{j}, j, \overline{j}$. Recall from Definition 5 the variables $\overline{x}^j = \sigma(x + e_j) = x + e_{\overline{j}}$ and $\underline{x}^j = \sigma(x - e_j) = x - e_{\underline{j}}$. Then, very similarly to how we derived (4.5) in Lemma 4.1.2, we get

$$\begin{aligned}
\sigma(\sigma(x - e_j) - e_i) &= \sigma(x - e_{\underline{j}} - e_i) \\
&= \begin{cases} x - e_{\underline{j}} - e_{\underline{i}} & \text{if } i > \overline{j} \\ x - e_{\underline{j}} - e_{\underline{i}} & \text{if } i < \underline{j} \\ x - e_{\underline{j}} - e_{\underline{j}+1} & \text{if } \underline{j} < i \leq \overline{j} \end{cases} \\
&= \begin{cases} x - e_{\underline{i}} - e_{\underline{j}} & \text{if } i > \overline{j} \text{ or } i < \underline{j} \\ x - e_{\underline{i}} - e_{\underline{j}+1} & \text{if } \underline{j} < i \leq \overline{j} \end{cases} \\
&= \begin{cases} \sigma(x^* - e_j) & \text{if } i > \overline{j} \text{ or } i < \underline{j} \\ \sigma(x^* - e_j) & \text{if } \underline{j} < i \leq \overline{j}, \underline{j} < j \\ \sigma(x^* - e_{j+1}) & \text{if } \underline{j} < i \leq \overline{j}, \underline{j} = j. \end{cases}
\end{aligned}$$

So for $i > \overline{j}$ or $i < \underline{j}$ as well as for $\underline{j} < i \leq \overline{j}, \underline{j} < j$ we can take $h = j$ to get (A.4). For $\underline{j} < i \leq \overline{j}, \underline{j} = j$ we can take $h = j + 1$, since $j < \overline{j}$. □

Using the Lemma above, we will use standard techniques to show how convexity of the relative value function V is usually proven. In our case this has not yet succeeded, but our attempts may serve as a basis for further research.

Conjecture A.1.1. *For convex f_1, it holds that V is convex, i.e.,*
$V(\sigma(x + e_j)) - 2V(x) + V(\sigma(x - e_j)) \geq 0$ for all $x \in \mathcal{X}$ such that $x_j \geq 1$, with $j = 1, \ldots, M$.

(Sketch of proof). We will use induction on n in V_n. Take $V_0(x) = 0$ for all x. Then obviously, $V_0(x)$ is convex for all $x \in \mathcal{X}$. Now assume that $V_n(x)$ is convex, i.e.,

$$V_n(\sigma(x + e_j)) - 2V_n(x) + V_n(\sigma(x - e_j)) \geq 0.$$

Then, for $n + 1$ we have the following.

$$\begin{aligned}
&V_{n+1}(\sigma(x + e_j)) - 2V_{n+1}(x) + V_{n+1}(\sigma(x - e_j)) = [f_1(\sigma(x + e_j)_M) - 2f_1(x_M) + f_1(\sigma(x - e_j)_M)] \\
&\underbrace{+ \mu \left(\sum_{i=1}^{M} \mathbb{I}\{(\sigma(x + e_j))_i > 0\} V_n(\sigma(\sigma(x + e_j) - e_i)) \right)}_{A_1} \\
&\underbrace{-2\mu \left(\sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} V_n(\sigma(x - e_i)) \right)}_{A_2} \underbrace{+ \mu \left(\sum_{i=1}^{M} \mathbb{I}\{(\sigma(x - e_j))_i > 0\} V_n(\sigma(\sigma(x - e_j) - e_i)) \right)}_{A_3} \\
&\underbrace{+ \left(1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{(\sigma(x + e_j))_i > 0\}) \right) V_n(\sigma(x + e_j))}_{B_1} \\
&\underbrace{-2 \left(1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\}) \right) V_n(x)}_{B_2} \underbrace{+ \left(1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{(\sigma(x - e_j))_i > 0\}) \right) V_n(\sigma(x - e_j))}_{B_3} \\
&\underbrace{+ \left[\min_{d \in \mathcal{D}} T_d^n(\sigma(x + e_j)) - 2 \min_{d \in \mathcal{D}} T_d^n(x) + \min_{d \in \mathcal{D}} T_d^n(\sigma(x - e_j)) \right]}_{C}.
\end{aligned}$$

Note that for convex f_1, the first term in square brackets is ≥ 0.

A_1, A_2 and A_3 can be dealt with as follows.

$$
\begin{aligned}
&A_1 + A_2 + A_3 = \\
&\mu\left(\sum_{i\neq\bar{j}} \mathbb{I}\{x_i>0\}V_n(\sigma(\sigma(x+e_j)-e_i)) + \underbrace{\mathbb{I}\{x_{\bar{j}}+1>0\}}_{=1}\underbrace{V_n(\sigma(\sigma(x+e_j)-e_{\bar{j}}))}_{=V_n(x)}\right)\\
&-2\mu\left(\sum_{i\neq j} \mathbb{I}\{x_i>0\}V_n(\sigma(x-e_i)) + \underbrace{\mathbb{I}\{x_j>0\}}_{=1}V_n(\sigma(x-e_j))\right)\\
&+\mu\left(\sum_{i\neq\underline{j}} \mathbb{I}\{x_i>0\}V_n(\sigma(\sigma(x-e_j)-e_i)) + \mathbb{I}\{x_{\underline{j}}-1>0\}V_n(\sigma(\sigma(x-e_j)-e_{\underline{j}}))\right)\\
&=\mu\left[\sum_{i\neq\underline{j},j,\bar{j}} \underbrace{\mathbb{I}\{x_i>0\}}_{\geq 0}\left(\underbrace{V_n(\sigma(\sigma(x+e_j)-e_i)) - 2V_n(\sigma(x-e_i)) + V_n(\sigma(\sigma(x-e_j)-e_i))}_{\geq 0 \text{ by the induction hypothesis and Lemma A.1.1}}\right)\right]\\
&+\mu\left[V_n(x) - 2V_n(\sigma(x-e_j)) + \mathbb{I}\{x_{\underline{j}}-1>0\}V_n(\sigma(\sigma(x-e_j)-e_{\underline{j}}))\right]\\
&+\mu\left[V_n(\sigma(\sigma(x+e_j)-e_j))\mathbb{I}\{j\neq\bar{j}\} + V_n(\sigma(\sigma(x+e_j)-e_{\underline{j}}))\mathbb{I}\{j\neq\underline{j}\}\right]\\
&-2\mu\left[V_n(\sigma(x-e_{\bar{j}}))\mathbb{I}\{j\neq\bar{j}\} + V_n(\sigma(x-e_{\underline{j}}))\mathbb{I}\{j\neq\underline{j}\}\right]\\
&+\mu\left[V_n(\sigma(\sigma(x-e_j)-e_{\bar{j}}))\mathbb{I}\{j\neq\bar{j}\} + V_n(\sigma(\sigma(x-e_j)-e_j))\mathbb{I}\{j\neq\underline{j}\}\right]\\
&\geq\mu\left[V_n(x) - 2V_n(\sigma(x-e_j)) + \mathbb{I}\{x_{\underline{j}}-1>0\}V_n(\sigma(\sigma(x-e_j)-e_{\underline{j}}))\right]\\
&+\mathbb{I}\{j\neq\bar{j}\}\mu\underbrace{\left[V_n(\sigma(\sigma(x+e_j)-e_j)) - 2V_n(\sigma(x-e_{\bar{j}})) + V_n(\sigma(\sigma(x-e_j)-e_{\bar{j}}))\right]}_{\text{I}}\\
&+\mathbb{I}\{j\neq\underline{j}\}\mu\underbrace{\left[V_n(\sigma(\sigma(x+e_j)-e_{\underline{j}})) - 2V_n(\sigma(x-e_{\underline{j}})) + V_n(\sigma(\sigma(x-e_j)-e_j))\right]}_{\text{II}}.
\end{aligned}
$$

I and II are both ≥ 0. This can be seen as follows.
For I, remark that $\sigma(x - e_{\bar{j}}) = \sigma(x - e_j)$. Also,

$$\sigma(\sigma(x+e_j)-e_j) = \sigma(x+e_{\bar{j}}-e_j) = x+e_{\bar{j}}-e_{\underline{j}} = x-e_{\underline{j}}+e_{\bar{j}} = \sigma(x-e_{\underline{j}}+e_{\bar{j}}) = \sigma(\sigma(x-e_j)+e_{\bar{j}}),$$

where in the second equality, it has been used that the k^{**} from Lemma 4.1.1 is the same (namely $\underline{j}$) for x and $x + e_{\bar{j}}$, when subtracting j, because $j \neq \bar{j}$. In the third and fourth equalities it has also been used that $j \neq \bar{j}$ and that the k^* from Lemma 4.1.1 is the same (namely $\bar{j}$) for both x and $x - e_{\underline{j}}$, when subtracting j. Moreover, $x - e_{\underline{j}} + e_{\bar{j}} \in \mathcal{X}$ and thus $\sigma(x - e_{\underline{j}} + e_{\bar{j}}) = x - e_{\underline{j}} + e_{\bar{j}}$. Concluding,

$$\mathrm{I} = \left[V_n(\sigma(\underline{x}^j + e_{\bar{j}}))) - 2V_n(\underline{x}^j) + V_n(\sigma(\underline{x}^j - e_{\bar{j}}))\right],$$

with $\underline{x}^j = \sigma(x - e_j) \in \mathcal{X}$, as defined in Definition 5. So $I \geq 0$ by the induction hypothesis.

For II, note that $\sigma(x - e_{\underline{j}}) = \sigma(x - e_j)$. Also,

$$\sigma(\sigma(x+e_j)-e_{\underline{j}}) = \sigma(x+e_{\bar{j}}-e_{\underline{j}}) = x+e_{\bar{j}}-e_{\underline{j}} = x-e_{\underline{j}}+e_{\bar{j}} = \sigma(x-e_j)+e_{\bar{j}} = \sigma(\sigma(x-e_j)+e_j),$$

where in the second equality it has been used that the k^{**} from Lemma 4.1.1 is the same (namely $\underline{j}$) for both x and $x+e_{\bar{j}}$ when subtracting $e_{\underline{j}}$, since $j \neq \underline{j}$. In the third equality it has been used that $\underline{j}$ is also the k^{**} for x when subtracting j. In the last equality it has been used that the k^* from Lemma 4.1.1 is the same (namely $\bar{j}$) for x and $\sigma(x - e_j)$ when adding j. Hence,

$$\mathrm{II} = \left[V_n(\sigma(\underline{x}^j + e_j))) - 2V_n(\underline{x}^j) + V_n(\sigma(\underline{x}^j - e_j))\right],$$

which is ≥ 0 by the induction hypothesis, since $\underline{x}^j \in \mathcal{X}$.

Furthermore,

$$B = B_1 + B_2 + B_3 = \left(1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\})\right) \underbrace{(V_n(\sigma(x + e_j)) - 2V_n(x) + V_n(\sigma(x - e_j)))}_{\geq 0 \text{ by the induction hypothesis}}$$

$$\begin{aligned} &+ \mu \mathbb{I}\{x_{\underline{j}} = 1\} V_n(\sigma(x - e_j)) \\ &\geq \mu \mathbb{I}\{x_{\underline{j}} = 1\} V_n(\sigma(x - e_j)), \end{aligned}$$

where we have used that

$$\sum_{i=1}^{M} \mathbb{I}\{(\sigma(x-e_j))_i > 0\} = \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} - \mathbb{I}\{x_{\underline{j}} = 1\},$$

and that $\mathbb{I}\{(\sigma(x+e_j))_k > 0\} = \mathbb{I}\{x_k > 0\}$ for all $k \in \{1, \dots, M\}$, since we have assumed $x_j \geq 1$ and hence also $x_{\bar{j}} \geq 1$.

Distinguishing the following two cases yields convexity of the remaining terms from A and B.

Case 1: $x_{\underline{j}} > 1$
In this case

$$\begin{aligned} A &\geq \mu \left[V_n(x) - 2V_n(\sigma(x-e_{\underline{j}})) + V_n(\sigma(\sigma(x-e_j)-e_{\underline{j}})) \right] \\ &= \mu \left[V_n(\sigma(\underline{x}^j + e_{\underline{j}})) - 2V_n(\underline{x}^j) + V_n(\sigma(\underline{x}^j - e_{\underline{j}})) \right] \end{aligned}$$

which is ≥ 0 by the induction hypothesis. The remaining term for B vanishes, since $\mathbb{I}\{x_{\underline{j}} = 1\} = 0$. Hence $A + B \geq 0$.

Case 2: $x_{\underline{j}} = 1$
In this case

$$\begin{aligned} A + B \geq & \underbrace{\mu \left[V_n(x) - 2V_n(\sigma(x-e_j)) \right]}_{\text{from A}} + \underbrace{\mu V_n(\sigma(x-e_j))}_{\text{from B}} \\ &= \mu \left[V_n(x) - V_n(\sigma(x-e_j)) \right] \\ &= \mu \left[\underbrace{V_n(\sigma(\underline{x}^j + e_{\underline{j}})) - V_n(\underline{x}^j)}_{\geq 0 \text{ due to non-decreasingness}} \right], \end{aligned}$$

where we have used that $\sigma(\underline{x}^j + e_{\underline{j}}) = \sigma(x - e_{\underline{j}} + e_{\underline{j}}) = x$.

Recall $T_d^n(x)$ from Definition 6. For C it holds that

$$\begin{aligned} C &= \left[\min_{d \in \mathcal{D}} T_d^n(\sigma(x+e_j)) - 2 \min_{d \in \mathcal{D}} T_d^n(x) + \min_{d \in \mathcal{D}} T_d^n(\sigma(x-e_j)) \right] \\ &\geq T_{d_1^*}^n(\sigma(x+e_j)) - T_{d_1^*}^n(x) - \left[T_{d_2^*}^n(x) - T_{d_2^*}^n(\sigma(x-e_j)) \right], \end{aligned}$$

where $d_1^* \in \operatorname{argmin}_{d\in\mathcal{D}}\{T_d^n(\sigma(x+e_j))\}$ and $d_2^* \in \operatorname{argmin}_{d\in\mathcal{D}}\{T_d^n(\sigma(x-e_j))\}$. If $d_1^* = d_2^*$, we are done since we can use the induction hypothesis straight away. If $d_1^* \neq d_2^*$ we will need the concept of sub- and supermodularity of the relative value function to show convexity. *Submodularity* of a function $f : \mathbb{N}_0^M \to \mathbb{R}$ is generally defined as

$$f(x) + f(x + e_i + e_j) \leq f(x + e_i) + f(x + e_j),$$

for all $1 \leq i < j \leq M$ and all $x \in \mathbb{N}_0^M$ (see also [11]). In our case, submodularity of the relative value function V comes down to

$$V(\sigma(x - e_j)) + V(\sigma(x + e_i)) \leq V(\sigma(\sigma(x - e_j) + e_i)) + V(x),$$

for all $1 \leq i, j \leq M$ and all $x \in \mathcal{X}$ such that $x_j \geq 1$. For *supermodularity*, the inequality sign should be reversed. If a function is both sub- as well as supermodular, it is called modular.

If $d_1^* \neq d_2^*$, we have

$$\begin{aligned}
C = {} & T_{d_1^*}^n(\sigma(x + e_j)) - T_{d_1^*}^n(x) - \left[T_{d_2^*}^n(x) - T_{d_2^*}^n(\sigma(x - e_j))\right] = \\
& f_2(d_1^*) - f_2(d_1^*) + \lambda \sum_{i=1}^{M-d_1^*+1} k(M, i, d_1^*) \left(V_n(\sigma(\sigma(x + e_j) + e_i)) - V_n(\sigma(x + e_i))\right) \\
& - \left[f_2(d_2^*) - f_2(d_2^*) + \lambda \sum_{i=1}^{M-d_2^*+1} k(M, i, d_2^*) \left(V_n(\sigma(x + e_i)) - V_n(\sigma(\sigma(x - e_j) + e_i))\right)\right] \\
= {} & \lambda \sum_{i=1}^{M-d_1^*+1} k(M, i, d_1^*) \left(V_n(\sigma(\sigma(x + e_j) + e_i)) - V_n(\sigma(x + e_i))\right) \\
& - \lambda \sum_{i=1}^{M-d_2^*+1} k(M, i, d_2^*) \left(V_n(\sigma(x + e_i)) - V_n(\sigma(\sigma(x - e_j) + e_i))\right)
\end{aligned} \tag{A.5}$$

Now we will change something to the way we treated B. To this end, note that since $k(M, i, d)$ are probabilities and sum to 1, we can write in B instead of λ also

$\lambda \sum_{i=1}^{M-d+1} k(M,i,d)$ for a $d \in \{1,\dots,M\}$. Hence, we can do the following.

$$
\begin{aligned}
&A + B \geq \\
&\left(1 - (\lambda \sum_{i=1}^{M-d_2^*+1} k(M,i,d_2^*) + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\})\right) (V_n(\sigma(x+e_j)) - 2V_n(x) + V_n(\sigma(x-e_j))) \\
&= \left(1 - \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\}\right) (V_n(\sigma(x+e_j)) - 2V_n(x) + V_n(\sigma(x-e_j))) \\
&\quad - \lambda \sum_{i=1}^{M-d_2^*+1} k(M,i,d_2^*) \left(V_n(\sigma(x+e_j)) - 2V_n(x) + V_n(\sigma(x-e_j))\right) \\
&\geq - \lambda \sum_{i=1}^{M-d_2^*+1} k(M,i,d_2^*) \left(V_n(\sigma(x+e_j)) - 2V_n(x) + V_n(\sigma(x-e_j))\right). \qquad \text{(A.6)}
\end{aligned}
$$

Using (A.5) and (A.6) we can write

$$
\begin{aligned}
&A + B + C \geq \\
&\lambda \sum_{i=1}^{M-d_1^*+1} k(M,i,d_1^*) \left(V_n(\sigma(\sigma(x+e_j)+e_i)) - V_n(\sigma(x+e_i))\right) \\
&- \lambda \Bigg(\sum_{i=1}^{M-d_2^*+1} k(M,i,d_2^*)[V_n(\sigma(x+e_j)) - V_n(x) + \\
&+ \underbrace{V_n(\sigma(x-e_j)) + V_n(\sigma(x+e_i)) - V_n(\sigma(\sigma(x-e_j)+e_i)) - V_n(x)]}_{\leq 0 \text{ if V submodular}} \Bigg) \\
&\geq \lambda \sum_{i=1}^{M-d_1^*+1} k(M,i,d_1^*) \left(V_n(\sigma(\sigma(x+e_j)+e_i)) - V_n(\sigma(x+e_i))\right) \\
&- \lambda \underbrace{\sum_{i=1}^{M-d_2^*+1} k(M,i,d_2^*)}_{=1} \underbrace{(V_n(\sigma(x+e_j)) - V_n(x))}_{\text{independent of } i} \\
&= \lambda \sum_{i=1}^{M-d_1^*+1} k(M,i,d_1^*) \left(\underbrace{V_n(\sigma(\sigma(x+e_j)+e_i)) - V_n(\sigma(x+e_i)) - V_n(\sigma(x+e_j)) + V_n(x)}_{\geq 0 \text{ if } V \text{ supermodular}} \right).
\end{aligned}
$$

So if V is both sub- as well as supermodular (i.e., if V is modular), then also in the case that $d_1^* \neq d_2^*$ we have proven convexity. □

In order for the above proof to be relevant to us, we should prove that V is modular. However, Figure A.1 shows that V is neither super- nor submodular. Conclusively, the above arguments will not lead to a complete proof, although it does show the standard techniques usually employed to proof convexity of V.

x2=	5	492,5205	493,5129	496,8591	505,157	522,3493	554,4601
	4	454,143	455,1525	458,623	467,3856	485,9204	
	3	434,8549	435,8292	439,373	448,4094		
	2	426,436	427,3803	430,8097			
	1	423,497	424,3462				
	0	422,8094					
		x1= 0	1	2	3	4	5

Testing convexity horizontally:				**Testing submodularity**			
2,35372	4,951712	8,894313	14,91866	0,016995	0,124345	0,464662	1,342615
2,461069	5,292029	9,772266		-0,0352	0,073317	0,27377	
2,569585	5,492482			-0,03	-0,11444		
2,485141				-0,09503			
vertically:				**Testing supermodularity**			
19,0893	19,03711	18,98608	18,79519	-0,017	-0,12434	-0,46466	-1,34261
10,86927	10,87448	10,68672		0,035198	-0,07332	-0,27377	
5,479818	5,414783			0,029995	0,11444		
2,25149				0,095031			
So it looks convex.				**So it is not super- nor submodular.**			
Testing superconvexity horizontally:				**vertically:**			
2,478065	5,416374	10,23693		19,0541	19,11043	19,25985	
2,534387	5,565799			10,83928	10,76004		
2,455146				5,384787			
So it looks superconvex.							

Figure A.1: Testing structural properties for $M = 4, K = 10, w_1 = 5, w_2 = 1$, keeping variables x_3 and x_4 fixed to value 5.

Note that, apart from assuming $d_1^* \neq d_2^*$, we did not need to assume any other conditions on d_1^* and d_2^*. In what follows, we will distinghuish different cases in which either we assume $d_1^* < d_2^*$ or vice versa. In order to keep things as simple as possible, we will now consider the case where the number of servers, M, is equal to 2. Unfortunately, like for the general case, we have not yet been able to prove convexity

for this two-dimensional case either. So we again consider Conjecture A.1.1 and show that the assumptions we make, conflict with the results we get from Figure A.1. In particular, we avoid using sub- or supermodularity, since Figure A.1 already excludes these properties.
In this two-dimensional case, we will change notation slightly. A state is represented by $(x, y) \in \mathcal{X}$, where $x \leq y$. In order to show that the assumptions we have to make conflict with the results we get from tests, it suffices to assume that $x < y - 1$. This assumption, that ensures that we do not have to sort the state variables in the reasoning below, allows for a simpler analysis. Furthermore, we will only focus on term C in the proof of Conjecture A.1.1, since the other terms we got rid of in the general case already[1].

Proof. We will consider four different cases. Recall that in the induction hypothesis we assume that V_n is convex.

Case 1a): $d_1^* = 1, d_2^* = 2, j = 1$

$$\begin{aligned}
C =& \left[\min_{d\in\mathcal{D}} T_d^n(\sigma(x+1,y)) - 2\min_{d\in\mathcal{D}} T_d^n(x,y) + \min_{d\in\mathcal{D}} T_d^n(\sigma(x-1,y))\right] \\
=& \left(T_{d_1^*}^n(x+1,y) - T_{d_1^*}^n(x,y)\right) - \left(T_{d_2^*}^n(x,y) - T_{d_2^*}^n(x-1,y)\right) \\
=& \frac{1}{2}\left(V_n(x+2,y) - V_n(x+1,y)\right) + \frac{1}{2}\left(V_n(x+1,y+1) - V_n(x,y+1)\right) \\
& - \left(V_n(x+1,y) - V_n(x,y)\right) \\
=& \frac{1}{2}\left(-3V_n(x+1,y) + 2V_n(x,y) - V_n(x,y+1) + V_n(x+1,y+1) + V_n(x+2,y)\right) \\
\geq& \frac{1}{2}\left(V_n(x+1,y+1) + V_n(x,y) - V_n(x,y+1) - V_n(x+1,y)\right),
\end{aligned}$$

where in the last inequality convexity is used. That is

$$\frac{1}{2}V_n(x+2,y) - V_n(x+1,y) + \frac{1}{2}V_n(x,y) \geq 0.$$

Note, however, that we are left with supermodularity, of which we know that it does not hold.

[1] Of course it might be that we have to use the other terms in combination with term C in order to arrive at a proof. We have tried doing this too, but these attempts did not bring us very far. Therefore, we have omitted them here.

Case 1b): $d_1^* = 1, d_2^* = 2, j = 2$

$$\begin{aligned}
C =& \left[\min_{d\in\mathcal{D}} T_d^n(\sigma(x,y+1)) - 2\min_{d\in\mathcal{D}} T_d^n(x,y) + \min_{d\in\mathcal{D}} T_d^n(\sigma(x,y-1))\right] \\
=& \left(T_{d_1^*}^n(x,y+1) - T_{d_1^*}^n(x,y)\right) - \left(T_{d_2^*}^n(x,y) - T_{d_2^*}^n(x,y-1)\right) \\
=& \frac{1}{2}\left(V_n(x+1,y+1) - V_n(x+1,y)\right) + \frac{1}{2}\left(V_n(x,y+2) - V_n(x,y+1)\right) \\
& - \left(V_n(x+1,y) - V_n(x+1,y-1)\right) \\
=& \frac{1}{2}\left(-3V_n(x+1,y) + 2V_n(x+1,y-1) - V_n(x,y+1) + V_n(x+1,y+1) + V_n(x,y+2)\right) \\
\geq& \frac{1}{2}\left(-V_n(x+1,y) + V_n(x+1,y-1) - V_n(x,y+1) + V_n(x,y+2)\right),
\end{aligned}$$

which is not a standard structural property. In the last inequality, again convexity is used. That is, $\frac{1}{2}V_n(x+1,y+1) - V_n(x+1,y) + \frac{1}{2}V_n(x+1,y-1) \geq 0$.
So in the case where $d_1^* = 1, d_2^* = 2$, we would at least need supermodularity to prove convexity, but Figure A.1 shows that V is certainly not supermodular. Moreover, in the case where $j = 2$, we are left with terms that do not construct a structural property that is standard (see also [11]) and would thus need further investigation to check whether it holds.

Case 2a): $d_1^* = 2, d_2^* = 1, j = 1$

$$\begin{aligned}
C =& \left[\min_{d\in\mathcal{D}} T_d^n(\sigma(x+1,y)) - 2\min_{d\in\mathcal{D}} T_d^n(x,y) + \min_{d\in\mathcal{D}} T_d^n(\sigma(x-1,y))\right] \\
=& \left(T_{d_1^*}^n(x+1,y) - T_{d_1^*}^n(x,y)\right) - \left(T_{d_2^*}^n(x,y) - T_{d_2^*}^n(x-1,y)\right) \\
=& \left(V_n(x+2,y) - V_n(x+1,y)\right) \\
& - \left(\frac{1}{2}\left(V_n(x+1,y) - V_n(x,y)\right) + \left(V_n(x,y+1) - V_n(x-1,y+1)\right)\right) \\
=& \frac{1}{2}\left(-3V_n(x+1,y) + 2V_n(x+2,y) + V_n(x,y) - V_n(x,y+1) + V_n(x-1,y+1)\right) \\
\geq& \frac{1}{2}\left(V_n(x+2,y) - V_n(x+1,y) - V_n(x,y+1) + V_n(x-1,y+1)\right),
\end{aligned}$$

which is not a standard structural property. In the last inequality, convexity applied to $\frac{1}{2}V_n(x+2,y) - V_n(x+1,y) + \frac{1}{2}V_n(x,y)$ is used.

Case 2b): $d_1^* = 2, d_2^* = 1, j = 2$

$$\begin{aligned}
C =& \left[\min_{d\in\mathcal{D}} T_d^n(\sigma(x,y+1)) - 2\min_{d\in\mathcal{D}} T_d^n(x,y) + \min_{d\in\mathcal{D}} T_d^n(\sigma(x,y-1))\right] \\
=& \left(T_{d_1^*}^n(x,y+1) - T_{d_1^*}^n(x,y)\right) - \left(T_{d_2^*}^n(x,y) - T_{d_2^*}^n(x,y-1)\right) \\
=& \left(V_n(x+1,y+1) - V_n(x+1,y)\right) \\
& - \left(\frac{1}{2}\left(V_n(x+1,y) - V_n(x+1,y-1)\right) + \left(V_n(x,y+1) - V_n(x,y)\right)\right) \\
=& \frac{1}{2}\left(-3V_n(x+1,y) + 2V_n(x+1,y+1) + V_n(x+1,y-1) + V_n(x,y+1) + V_n(x,y)\right) \\
\geq& \frac{1}{2}\left(V_n(x+1,y+1) + V_n(x,y) - V_n(x,y+1) - V_n(x+1,y)\right),
\end{aligned}$$

where in the last inequality convexity is used. That is

$$\frac{1}{2}V_n(x+1,y+1) - V_n(x+1,y) + \frac{1}{2}V_n(x+1,y-1) \geq 0.$$

Note, however, that we are again left with supermodularity.
Conclusively, in both cases we would need supermodularity to prove convexity, which conflicts with Figure A.1. Hence, this approach will not give us the desired proof of convexity either. □

Finally, we have yet another different approach. Firstly, note that SQ(2) is, in the two-dimensional case, equivalent to JSQ. Since we study the dynamic version of SQ(d), in the optimality equation there will also be terms corresponding to JSQ. So, in this two-dimensional case, we can approach the problem slightly differently than before, using the fact that the relative value function of JSQ is convex. Next to terms corresponding to JSQ, also terms that correspond to suboptimal transitions of SQ(1) may be present. In the analysis below we have indicated to which of both the terms involved refer.

(Sketch of proof). Before we start with the analysis, we should, next to non-decreasingness, convexity and sub- and supermodularity, first introduce two other structural properties. *Superconvexity* of a function $f : \mathbb{N}_0^M \to \mathbb{R}$ is generally defined as

$$f(x + e_i) + f(x + e_i + e_j) \leq f(x + e_j) + f(x + 2e_i),$$

for all $1 \leq i, j \leq M, i \neq j$ and all $x \in \mathbb{N}_0^M$ (see also [11]). For *superconcavity*, the inequality sign should be reversed. Now we distinghuish again the four different cases

we have considered above.

Case 1a): $d_1^* = 1, d_2^* = 2, j = 1$

$$
\begin{aligned}
C &= \left[\min_{d\in\mathcal{D}} T_d^n(\sigma(x+1,y)) - 2\min_{d\in\mathcal{D}} T_d^n(x,y) + \min_{d\in\mathcal{D}} T_d^n(\sigma(x-1,y))\right] \\
&= \left(T_{d_1^*}^n(x+1,y) - T_{d_1^*}^n(x,y)\right) - \left(T_{d_2^*}^n(x,y) - T_{d_2^*}^n(x-1,y)\right) \\
&= \frac{1}{2}\underbrace{\left(V_n(x+2,y) - V_n(x+1,y)\right)}_{\text{JSQ}} + \frac{1}{2}\underbrace{\left(V_n(x+1,y+1) - V_n(x,y+1)\right)}_{\text{suboptimal}} \\
&\quad - \underbrace{\left(V_n(x+1,y) - V_n(x,y)\right)}_{\text{JSQ}} \\
&\quad + \underbrace{\frac{1}{2}\left(V_n(x+2,y) - V_n(x+1,y)\right) - \frac{1}{2}\left(V_n(x+2,y) - V_n(x+1,y)\right)}_{=0 \text{ (dummy term)}} \\
&= \underbrace{V_n(x+2,y) - 2V_n(x+1,y) + V_n(x,y)}_{\geq 0 \text{ by convexity}} \\
&\quad + \underbrace{\frac{1}{2}\left(V_n(x+1,y) + V_n(x+1,y+1) - V_n(x+2,y) - V_n(x,y+1)\right)}_{\geq 0 \text{ if } V \text{ is superconcave}} \\
&\geq 0
\end{aligned}
$$

Case 1b): $d_1^* = 1, d_2^* = 2, j = 2$

$$
\begin{aligned}
C &= \left[\min_{d\in\mathcal{D}} T_d^n(\sigma(x,y+1)) - 2\min_{d\in\mathcal{D}} T_d^n(x,y) + \min_{d\in\mathcal{D}} T_d^n(\sigma(x,y-1))\right] \\
&= \Big(T_{d_1^*}^n(x,y+1) - T_{d_1^*}^n(x,y)\Big) - \Big(T_{d_2^*}^n(x,y) - T_{d_2^*}^n(x,y-1)\Big) \\
&= \frac{1}{2}\underbrace{(V_n(x+1,y+1) - V_n(x+1,y))}_{\text{JSQ}} + \frac{1}{2}\underbrace{(V_n(x,y+2) - V_n(x,y+1))}_{\text{suboptimal}} \\
&\quad - \underbrace{(V_n(x+1,y) - V_n(x+1,y-1))}_{\text{JSQ}} \\
&\quad + \underbrace{\frac{1}{2}(V_n(x+1,y+1) - V_n(x+1,y)) - \frac{1}{2}(V_n(x+1,y+1) - V_n(x+1,y))}_{=0 \text{ (dummy term)}} \\
&= \underbrace{V_n(x+1,y+1) - 2V_n(x+1,y) + V_n(x+1,y-1)}_{\geq 0 \text{ by convexity}} \\
&\quad + \underbrace{\frac{1}{2}(V_n(x,y+2) + V_n(x+1,y) - V_n(x,y+1) - V_n(x+1,y+1))}_{\geq 0 \text{ if superconvex}} \\
&\geq 0.
\end{aligned}
$$

So in the case where $d_1^* = 1, d_2^* = 2$, we find that we need two conflicting (in the strict sense) properties, namely superconcavity and superconvexity. Moreover, Figure A.1 shows that V is certainly not strictly superconcave, but probably superconvex.

Case 2a): $d_1^* = 2, d_2^* = 1, j = 1$

$$
\begin{aligned}
C =& \left[\min_{d\in\mathcal{D}} T_d^n(\sigma(x+1,y)) - 2\min_{d\in\mathcal{D}} T_d^n(x,y) + \min_{d\in\mathcal{D}} T_d^n(\sigma(x-1,y))\right] \\
=& \left(T_{d_1^*}^n(x+1,y) - T_{d_1^*}^n(x,y)\right) - \left(T_{d_2^*}^n(x,y) - T_{d_2^*}^n(x-1,y)\right) \\
=& \underbrace{(V_n(x+2,y) - V_n(x+1,y))}_{\text{JSQ}} \\
& - \left(\frac{1}{2}\underbrace{(V_n(x+1,y) - V_n(x,y))}_{\text{JSQ}} + \underbrace{(V_n(x,y+1) - V_n(x-1,y+1))}_{\text{suboptimal}}\right) \\
& - \underbrace{\frac{1}{2}(V_n(x+1,y) - V_n(x,y)) + \frac{1}{2}(V_n(x+1,y) - V_n(x,y))}_{=0 \text{ (dummy term)}} \\
=& \underbrace{V_n(x+2,y) - 2V_n(x+1,y) + V_n(x,y)}_{\geq 0 \text{ by convexity}} \\
& + \underbrace{\frac{1}{2}(V_n(x,y+1) + V_n(x+1,y) - V_n(x-1,y+1) - V_n(x,y))}_{\geq 0 \text{ by applying non-decreasingness twice}} \\
\geq& 0.
\end{aligned}
$$

Case 2b): $d_1^* = 2, d_2^* = 1, j = 2$

$$
\begin{aligned}
C &= \left[\min_{d\in\mathcal{D}} T_d^n(\sigma(x,y+1)) - 2\min_{d\in\mathcal{D}} T_d^n(x,y) + \min_{d\in\mathcal{D}} T_d^n(\sigma(x,y-1))\right] \\
&= \Big(T_{d_1^*}^n(x,y+1) - T_{d_1^*}^n(x,y)\Big) - \Big(T_{d_2^*}^n(x,y) - T_{d_2^*}^n(x,y-1)\Big) \\
&= \underbrace{(V_n(x+1,y+1) - V_n(x+1,y))}_{\text{JSQ}} \\
&\quad - \left(\frac{1}{2}\underbrace{(V_n(x+1,y) - V_n(x+1,y-1))}_{\text{JSQ}} + \underbrace{(V_n(x,y+1) - V_n(x,y))}_{\text{suboptimal}}\right) \\
&\quad \underbrace{- \frac{1}{2}(V_n(x+1,y) - V_n(x+1,y-1)) + \frac{1}{2}(V_n(x+1,y) - V_n(x+1,y-1))}_{=0 \text{ (dummy term)}} \\
&= \underbrace{V_n(x+1,y+1) - 2V_n(x+1,y) + V_n(x+1,y-1)}_{\geq 0 \text{ by convexity}} \\
&\quad + \underbrace{\frac{1}{2}(V_n(x,y) + V_n(x+1,y) - V_n(x,y+1) - V_n(x+1,y-1))}_{\geq 0 \text{ if superconcave}} \\
&\geq 0.
\end{aligned}
$$

So also in this second case we need superconcavity of V to prove convexity. However, as mentioned before, Figure A.1 shows that V is certainly not superconcave.
Conclusively, we can say that this approach will not give us the desired proof of convexity either. □

Properties of d_1^*, d_2^* and $k(M,i,d)$

Recall $d_1^* \in \text{argmin}_{d\in\mathcal{D}}\{T_d^n(\sigma(x+e_j))\}$ and $d_2^* \in \text{argmin}_{d\in\mathcal{D}}\{T_d^n(\sigma(x-e_j))\}$, for a $j = 1,\ldots,M$. In the proofs we have considered above, we distinguished between different cases with regard to d_1^* and d_2^*. That is, we assumed $d_1^* < d_2^*$ or vice versa. In might be useful to be able to prove that only one of them holds. In order to possibly arrive at such properties, it is worthwhile studying $k(M,i,d)$. Recall $k(M,i,d) = d\frac{(M-i)!(M-d)!}{(M-i-d+1)!M!}$ in the minimizing expression of the optimality equation (3.15). However, doing a simple test for $N = 2$ and truncation level $K = 7$ gives the result in Figure A.2, and shows that both $d_1^* < d_2^*$ and $d_1^* > d_2^*$ as well as $d_1^* = d_2^*$

might occur.
Nevertheless, it might be worthwhile looking into properties of $k(M,i,d)$ anyway. It can be shown that for fixed i and M, $k(M,i,d)$ is increasing in d for $d \leq \lceil \frac{M-i+1}{i} \rceil$ and decreasing for $d > \lceil \frac{M-i+1}{i} \rceil$. It can also be shown that $k(M,i,d)/d = \frac{(M-i)!(M-d)!}{(M-i-d+1)!M!}$ is decreasing in d. We have summarized these properties in the lemma below. See also Figure A.3. Due to the non-linear and non-monotone structure of $k(M,i,d)$ we have not been able to exploit these properties.

Lemma A.1.2. *Given $M \geq 1$ and $i = 1, \ldots, M$, the following holds:*

1. *The function $k(M,i,d) := d\frac{(M-i)!(M-d)!}{(M-i-d+1)!M!}$ is increasing in d for $d \leq \lceil \frac{M-i+1}{i} \rceil$ and decreasing for $d > \lceil \frac{M-i+1}{i} \rceil$, with $d \leq M-i+1$.*
2. *The function $k(M,i,d)/d$ is decreasing in d, with $d \leq M-i+1$.*

Proof. For the first part, we would like to investigate the difference $k(M,i,d+1) - k(M,i,d)$. For $k(M,i,d+1)$ to be larger than 0 we must assume $d \leq M-1$ and $i \leq M-d$. Then,

$$\begin{aligned}
k(M,i,d+1) - k(M,i,d) =& \frac{(M-i)!}{M!}\left((d+1)\frac{(M-d-1)!}{(M-i-d)!} - d\frac{(M-d)!}{(M-i-d+1)!}\right) \\
=& \frac{(M-i)!}{M!}\left((d+1)\frac{(M-d-1)!}{(M-i-d)!} - d\frac{(M-d)(M-d-1)!}{(M-i-d+1)(M-i-d)!}\right) \\
=& \underbrace{\frac{(M-i)!(M-d-1)!}{M!(M-i-d)!}}_{A}\underbrace{\left((d+1) - d\frac{(M-d)}{(M-i-d+1)}\right)}_{B}.
\end{aligned}$$

Note that for $d \leq M-1$ and $i \leq M-d$ it holds that $A \geq 0$. Consequently,

$$\begin{aligned}
k(M,i,d+1) - k(M,i,d) \geq 0 &\iff B \geq 0 \\
&\iff (d+1) - d\frac{(M-d)}{(M-i-d+1)} \geq 0 \\
&\iff (d+1)(M-i-d+1) \geq d(M-d) \\
&\iff M-i+1 \geq di \\
&\iff d \leq \frac{M-i+1}{i}.
\end{aligned}$$

Since d must be an integer $1 \leq d \leq M-1$, we have that $k(M,i,d)$ is increasing for $d \leq \left\lceil \frac{M-i+1}{i} \right\rceil$ and decreasing for $d > \left\lceil \frac{M-i+1}{i} \right\rceil$, where it must be noted that if $\left\lceil \frac{M-i+1}{i} \right\rceil = 1$, it is strictly decreasing.
For the second part we look at

$$\begin{aligned}\frac{k(M,i,d+1)}{d+1} - \frac{k(M,i,d)}{d} =& \frac{(M-i)!}{M!}\left(\frac{(M-d-1)!}{(M-i-d)!} - \frac{(M-d)!}{(M-i-d+1)!}\right) \\ =& \frac{(M-i)!}{M!}\left(\frac{(M-d-1)!(M-i-d+1)-(M-d)!}{(M-i-d+1)!}\right) \\ \leq& 0,\end{aligned}$$

since $(M-d-1)!(M-i-d+1) \leq (M-d)!$ and all other terms involved are ≥ 0.
So $k(M,i,d)/d$ is decreasing in d. □

A.2 A Markov reward process for the Round Robin algorithm

For the sake of completeness, we show in this appendix how to formulate a Markov reward process for the Round Robin (RR) algorithm, despite not being very practical for high dimensional systems.
Recall that the RR algorithm does not need any information about the number of jobs per subsystem, but just assigns jobs sequentially to all subsystem and repeats this when all subsystems have been given a job.
Let $\mathcal{X} = (\mathbb{N}_0)^M$ be the state space. For $x \in \mathcal{X}$, x_i represents the number of jobs in the subsystem that received a package i arrivals ago. According to the RR algorithm, an arriving job had to be assigned to subsystem M. In order to keep this representation of x_i intact, we need the following function that makes sure that after an arrival all x_i, $i = 1, \ldots, M-1$ shift one position to the right and x_M becomes x_1. That is, define $\gamma : \mathcal{X} \to \mathcal{X}$ as $\gamma(x) = \tilde{x}$ with $\tilde{x}_i = x_{i-1}, i = 2, \ldots, M$, and $\tilde{x}_1 = x_M$. So if we apply this function to x after an arrival, then indeed x_i represents the number of jobs in the subsystem that received a package i arrivals ago. Using γ, we can specify the transition probabilities.

$$p(x,x') = \begin{cases} \lambda & \text{if } x' = \gamma(x + e_M) \\ \mu \mathbb{I}\{x_i > 0\} & \text{if } x' = x - e_i \\ 1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\}) & \text{if } x' = x \\ 0 & \text{otherwise.} \end{cases} \tag{A.7}$$

Since we want to minimize the largest queue length, we take as cost function $c(x) = \max_{i=1,\dots,M}\{x_i\}$. The corresponding Poisson equation is

$$V(x) + g = c(x) + \lambda V(\gamma(x + e_M)) \tag{A.8}$$

$$+ \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\} V(x - e_i) \tag{A.9}$$

$$+ \left[1 - (\lambda + \mu \sum_{i=1}^{M} \mathbb{I}\{x_i > 0\})\right] V(x). \tag{A.10}$$

A.3 Simulation

In Chapter 5 we used simulation to compare performance of different algorithms. In this appendix we will elaborate a bit more on the specifics of the simulations we have done.

Firstly, we have used simulation using the method of batch means. That is, for each algorithm we simulated $N = 200000$ events (an event is either an arrival or a departure) of which the first $n_0 = 50000$ were not taken into account (warm-up period) and the rest is divided over $m = 30$ batches consisting of $n = 5000$ events each. In each batch, the maximum queue length[2] is recorded at each event and the result is time-averaged. This results in 30 so-called *batch means* $\bar{x}_i$, $i = 1, \dots, m$. We checked whether the size of the warm-up period and the batches we took were big enough, using the theory in [8]. To this end we had to look at the variance of $\bar{x} = (\bar{x}_1, \dots, \bar{x}_m)$,

$$\mathrm{Var}(\bar{x}) := \frac{1}{m-1} \sum_{i=1}^{m} (\bar{x}_i - \bar{\bar{x}})^2,$$

where $\bar{\bar{x}} := \frac{1}{m} \sum_{i=1}^{m} \bar{x}_i$, and the autocovariance between two consecutive batch means,

$$\mathrm{Cov}(\bar{x}_i, \bar{x}_{i+1}) := \frac{1}{m-2} \sum_{i=1}^{m-1} (\bar{x}_i - \bar{\bar{x}})(\bar{x}_{i+1} - \bar{\bar{x}}).$$

This is illustrated for the SQ(2) algorithm for $M = 4$, $w_1 = 5$ and $\rho = 0.8$ in Figure A.4 and Table A.1. In hindsight it seems that a warm-up period of 50000 is definitely long enough, but that the batch size could have been taken longer. However this

[2]Recall that with maximum queue length we mean the job in service plus the jobs waiting in the queue.

would also have entailed simulating more events N, because of a subsequent decrease in the number of batches m. That is, if m is too low, the corresponding confidence interval would become too wide. Although this brings about longer computation times, it is something that should be taken into account in future research.
For the values we have specified above, the corresponding 95% confidence interval is generally within 3% of the point estimates.

Batch size n	Autocovariance N=150 000	Variance N=150 000	Autocovariance N=1 000 000	Variance N=1 000 000
1	52.5508	54.5760	55.6153	57.6285
2	64.9076	67.5144	55.3310	57.9171
5	54.3898	58.5357	52.6441	56.8291
10	42.9124	49.3942	45.5388	51.9287
50	34.1890	49.5394	26.5533	41.9205
100	19.1208	39.5084	14.9437	34.1000
200	6.0185	23.9124	7.7577	26.7631
500	2.2055	14.1991	2.3615	14.9272
1000	0.6065	8.2286	0.6978	6.8537
2000	-0.1580	2.9695	-0.0378	4.7194
2500	-0.2641	2.6083	0.0878	2.9222
5000	0.3267	1.3477	0.0738	1.5612
10000	0.1122	0.6932	0.0481	0.8002
25000	-0.3380	0.4564	0.0138	0.3835

Table A.1: Given are the autocovariance and variance of simulations of SQ(2), with $M = 4$, $w_1 = 5$ and $\rho = 0.8$. We compare $N = 200000$ to $N = 1000000$ for various batch sizes, with $n_0 = 50000$. In [8] it is explained that the batch size n should be such that the autocovariance of the batch means is small compared to their variance. In our simulations we computed $N = 200000$ events and took a batch size of $n = 5000$. In the corresponding columns of this table, one might notice that taking $N = 1000000$ and $n = 25000$ might have been better. On the other hand, this would have increased computation times also considerably.

A.4 Matlab code

Below you find the Matlab code of the value iteration algorithm for the MDP that models the optimal dynamic SQ(d) policy. If you are interested in any of the other

codes I wrote, please send me an email and I will provide you with it.

```
%This m-file performs value iteration for the MDP modeling a dynamic SQ(d)
%policy in a system with N queues %with Poisson arrivals with rate L, and
%exponential service times with rate M, at every subsystem. The state of
%the system keeps track of the number of customers in every subsystem,
%ordering them from small to big.

function [g policy VVV iter it V] = MDPsqdpolicy (N,K,L,M,w1,w2,dmax,epsilon)
%N=number of queues, K=maximum number of customers per subsystem (truncation
%level), %L=arrival rate, M=service rate per server (i.e., maximum service
%rate is N*M), N should be greater than 1.
%dmax=maximum d allowed (dmax<=N should hold), w1=weighing factor of regular
%costs, w2=weighing factor for increasing d (communication costs).

L2 = L;
M2 = M;
L = L2/(L2+N*M2);       %scaling arrival and service rates such that L+N*M=1
M = M2/(L2+N*M2);

v = 0:K;                %v is the vector [0 1 2 ... K] which we need as input
%for 'combsrep.m'

X = combsrep(v,N);      %matrix of all possible x (i.e., all x such that
%x1<=x2<=...<=xN holds)

A = length(X(:,1));     %total number of possible states x

dim=repmat(K+1,[1 N]);  %create the dimension of the multidimensional array
                        %it should have dimension (K+1)^N

V = zeros(dim);         %allocate memory for value function V
VVV = zeros(dim);       %The dimension of VVV should equal the dimension of
%V in order to be able to run the iteration
                        %later on. We have to allocate memory for VVV as
                        %well.

poli = zeros(A,1);      %in poli we will save the optimal policy
```

```
max2 = 100;             %we have to start with large enough max2 and min2 in
min2 = 0;               %order for the while loop below to start (we assume
%epsilon small)

while   max2-min2 > epsilon;    %value iteration

max2 = -10^10;          %have to reset max2 and min2 at every
min2 = 10^10;           %iteration in order for the last two if-loops
         %to make sense

VV = V;                 %VV is needed to run the iteration below

for h = 1:A             %for all possible x
x = X(h,:);          %each row of X is a possible x, we loop through them here.
x = ones(1,N)+x;     %we increase all states x with 1 in order to be able to use
%V(x) below
y = linindex(x,N,K); %y is the linear index corresponding to component
%values of x

V(y) = w1*(x(N)-1); %costs have the double structure c(x,d)=w1*(x(N)-1)+w2*d
%(the -1 is due to our increasing all x with 1 for algorithmic reasons).
                    %This is only the first part of the costs.
                    %The second part is added below, after minimization over d.

for l = 1:N         %sum over all N queues to get the part without
                    %decisions involved, representing departures
vol = (x(l)>1);
V(y) = V(y)+M*vol*VV(linindex(sortee(x,l),N,K))-M*vol*VV(y);
end

V(y)=V(y)+(1-L)*VV(y);  %add the last term that does not involve decisions

dee=zeros(1,dmax);
%below we add the terms to the value function in which
%decisions are involved.
for d=1:dmax        %for all possible decisions
maa=0;
```

```
for l = 1:N-d+1     %calculate the sum representing arrivals for the given d
maa=maa+((d*prod1(N,l,d))/(prod1(N,0,d+1)))*VV(linindex(sorteerK(x,l,K),N,K));
end
maa=L*maa+w2*d; %multiplying by L and adding the second part of the costs,
                %which depends on the decision chosen

dee(d)=maa;     %saving the outcome for every d in vector dee
end

[mini aant]=min(dee);   %take the minimum over d and remember the
                        %argmin (='aant'), i.e., which 'd' was optimal

V(y)=V(y)+mini;         %by adding the minimization term 'mini', the value
%function is complete

poli(h)=aant; %save the argmin to keep track of the decisions
%for every x

VVV(y)=aant;            %in 'VVV' we save the optimal decision per state

if  V(y)-VV(y) < min2;  %update min2 and max2 for the while loop
min2 = V(y)-VV(y);
end

if  V(y)-VV(y) > max2;
max2 = V(y)-VV(y);
end
end
end
g =(min2+max2)/2;   %here we use the formula V(x)-VV(x)-->tau*g
                    %to compute g, with tau=1 since we scaled
                    %lambda and mu such that L+N*M=1

policy=[X poli];    %for every state x, you find the optimal d
end
```

x1	x2	d*
0	0	1
0	1	2
0	2	2
0	3	2
0	4	2
0	5	2
0	6	2
1	1	1
1	2	2
1	3	2
1	4	2
1	5	2
1	6	2
2	2	1
2	3	2
2	4	2
2	5	2
2	6	2
3	3	1
3	4	2
3	5	2
3	6	2
4	4	1
4	5	2
4	6	2
5	5	1
5	6	2
6	6	1

Figure A.2: The optimal policy of the MDP with $M = 2, w_1 = 5, w_2 = 1, \lambda = 0.5, \mu = 1$ and the truncation level $K = 7$. Here it can be seen that both $d_1^* < d_2^*$ and $d_1^* > d_2^*$ as well as $d_1^* = d_2^*$ might occur. For example, for $(x, y) = (5, 5)$ we get with $j = 1$ that $d_1^* = d_2^* = 2$, when looking at $\sigma(x-1, y) = (4, 5)$ and $\sigma(x+1, y) = (5, 6)$. Similarly we get $1 = d_1^* < d_2^* = 2$ for e.g., $(x, y) = (4, 5)$ and $2 = d_1^* > d_2^* = 1$ for $(x, y) = (3, 4)$.

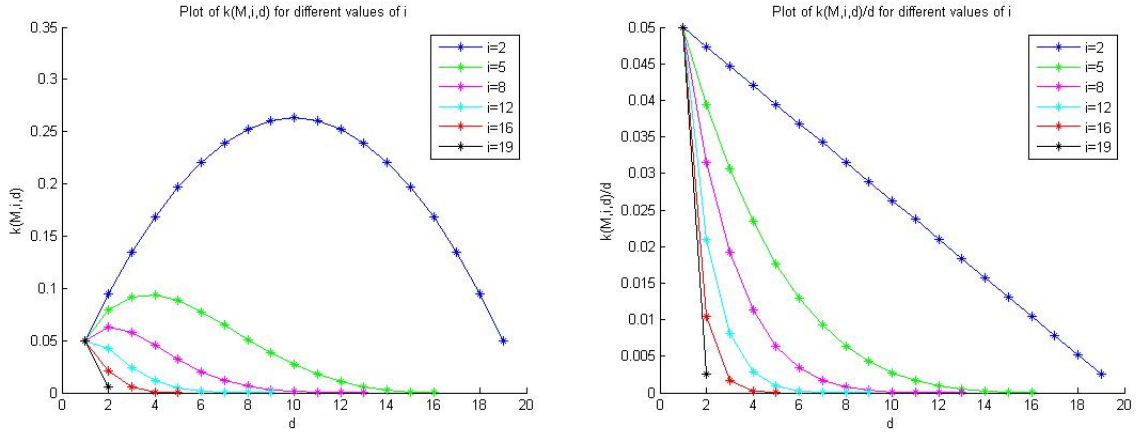


Figure A.3: On the left $k(M, i, d)$ is plotted for several i. It is increasing for $d \leq \left\lceil \frac{M-i+1}{i} \right\rceil$ and decreasing for $d > \left\lceil \frac{M-i+1}{i} \right\rceil$, where it must be noted that if $\left\lceil \frac{M-i+1}{i} \right\rceil = 1$, it is strictly decreasing. On the right $k(M, i, d)/d$ is plotted for several i and it is decreasing for all i.

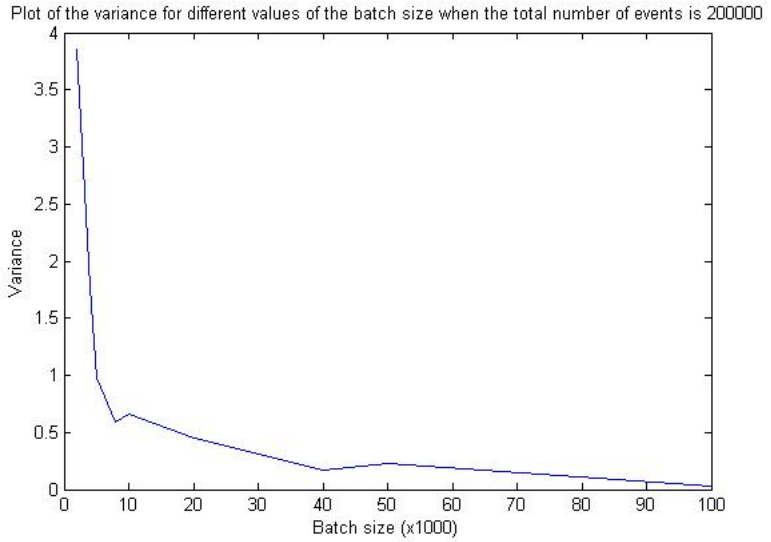


Figure A.4: The variance of a simulation of SQ(2) with $N = 200000$ events, for varying batch sizes, for $M = 4$, $w_1 = 5$ and $\rho = 0.8$. The length of the warm-up period (or *transient interval*) is the batch size at which the variance definitely starts decreasing. It seems that taking 50000 as warm-up period is more than sufficient.