Timeslot allocation for waiting list control

Tactical planning of orthopaedic surgeons at the Sint Maartenskliniek

Yanna van der Vlugt

TU Delft

Cover image: Anna Schvetz (2020). *Photo of Medical Professionals Wearing Personal Protective Equipment*.

Timeslot allocation for waiting list control

Tactical planning of orthopaedic surgeons at the Sint Maartenskliniek

by

Yanna van der Vlugt

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on Tuesday October 26, 2021 at 10:00 AM.

Student number: 4352459
Project duration: February 1, 2021 – October 26, 2021
Thesis committee:

Dr. ir. L.J.J. van Iersel,	TU Delft, chair
Dr. ir. J.T. van Essen,	TU Delft, supervisor
Dr. J.W. Böhmer,	TU Delft

TU Delft

Abstract

Patients visiting a hospital for elective surgery often have multiple consultations with a surgeon before undergoing surgery. Hospitals discern between different types of consultations, and make a schedule allocating timeslots of outpatient department sessions to these different consultation types several weeks in advance. Changing the proportion of consultation types affects the patient waiting lists for both consultations and surgery. However, the precise consequences of such interventions are uncertain, as not all patients follow the same treatment pathway. Furthermore, as these planning decisions are made far in advance, they are based on an uncertain prediction of future waiting lists. The goal is to use these interventions to control waiting lists, in order to reduce waiting times for patients and ensure that all available time capacity in the outpatient department and operating room is used. This problem is referred to as the *timeslot allocation problem*. In this thesis, we study the performance of various solution methods in achieving this goal.

The problem is modelled as a Markov decision process (MDP). As the state space is very large, the problem does not admit an exact solution. Therefore, least-squares policy iteration is used to find an approximate solution. We also formulate an (integer) linear program which is used to solve a deterministic variant of the MDP, and investigate some simple decision rules.

This thesis features a case study at the Sint Maartenskliniek, a hospital focusing on orthopaedic care in Nijmegen, the Netherlands. Data from the hospital is used to make a simulation with which solution methods can be tested and compared. We find that all methods improve on the static roster method used by the hospital, with the linear program leading to the best results. Furthermore, planning less far ahead allows for a better prediction of the state for which to plan, and so also leads to better performance. In the case of SMK, we recommend fixing 60% of the timeslots using a static roster method 12 weeks in advance, and using the integer linear program to schedule the remaining 40% of appointments 6 weeks in advance.

Preface

The thesis in front of you is the final project of my master's degree in Applied Mathematics, concluding my seven years as a student at the Delft University of Technology. I feel very lucky to have learned so much, to have met so many new people and to have faced so many exciting experiences and challenges along the way. In this preface, I would like to thank those that have helped me throughout this project.

I was very happy to have Theresia as my supervisor; thank you for all of your feedback and guidance. I enjoyed our discussions on Oxford commas and health care/healthcare/health-care during our weekly meetings. Most of all, I appreciate that you not only showed interest in my work, but also in my personal well-being and motivation levels. To Mijke, Rob and Olivia: thank you for your supervision on behalf of Rhythm and the Sint Maartenskliniek. I learned a lot from you about how mathematics is applied in the 'real world' and enjoyed getting to know the world of SMK and healthcare capacity planning. Finally, I want to thank Wendelin Böhmer for taking the time to provide me with information and advice on LSPI and reinforcement learning, and involving me in the RL community.

A lot of appreciation goes to my friends, housemates and family for supporting and motivating me throughout my studies. In particular, I would like to thank Rik for proof-reading my thesis, as well as Thomas and Joyce for our many study sessions. Working from home wasn't always easy, but you kept me sane, focused and well-caffeinated throughout the year.

Yanna van der Vlugt
Rotterdam, October 2021

Management summary

The Sint Maartenskliniek is a hospital group devoted to the treatment of medical conditions related to posture and movement. This thesis features a case study of healthcare capacity planning for the orthopaedic department of the hospital. Here, we provide a summary of this thesis detailing the most important findings and recommendations relevant to the Sint Maartenskliniek.

The planning intervention studied in this thesis is changing the number of first, repeat, and discharge consultations scheduled in a surgeon's outpatient department sessions during a planning period of two weeks. Our aim was to use this intervention to ensure that patient waiting times are within Dutch and self-imposed norms, while ensuring that appointment slots do not go unused due to too short waiting lists. The problem was formulated as a mathematical model, and a simulation was programmed using data from the Sint Maartenskliniek in order to test and compare a variety of solution methods.

Waiting lists for appointment types are modelled as queues. Each queue has one or more urgency levels, indicating the deadline before which a patient must be treated. For example, the repeat consultation queue has three urgency levels, determined by the hospital: six weeks, three months, and six months. The first consultation queue has one urgency level of four weeks, as defined by the Treeknorm on external access times. We define the state of the system by the number of patients of each urgency level, and their waiting times.

Since the schedules for surgeons are made twelve weeks in advance, a prediction must be made of the expected state of the system in twelve weeks time. In this thesis, we develop a method to do so. The prediction is based on the current state, the activity rosters in the upcoming weeks, and system parameters such as transition probabilities between queues and the expected number of new patients in each two-week period. We find that the difference between the realized and predicted state increases linearly when the amount of time to plan ahead for is increased, with approximately 20% of patients being in different queues than expected when planning twelve weeks ahead.

The solution methods must determine how many patients from each queue to treat, depending on the number of available timeslots in the outpatient department and OR, the expected state of the system, and system parameters such as transition probabilities and the expected number of new patients. The objective is defined as minimizing a cost incurred by patients who are waiting longer than permitted by their urgency level, and maximizing a reward gained by treating patients. The relative values of these costs and rewards impact behaviour of the solution methods (i.e., the type of appointments prioritized), and should be chosen carefully, for example by using a simulation to test the effects of changing these parameters on performance measures.

The first solution method investigated in this thesis is least-squares policy iteration, a rather complex algorithm which provides an approximation to the optimal solution of the mathematical model. We also formulated an (integer) linear program ((I)LP), which uses transition fractions rather than probabilities, in order to remove stochasticity from the problem. Both of these methods take the future effects of current decisions into account. Decision rules were also formulated, which are much easier to understand and implement, but can only optimize for the current time period. These methods are compared to the method used by SMK, which is using static (fixed) rosters.

We obtain the best results, in terms of the contribution function, from the (I)LP. This method does require the use of some form of optimization software, such as AIMMS or Gurobi. If this software is not available, the Highest Contribution decision rule, which selects patients to allocate slots for in order of highest cost + reward gained by treating that patient, can be used as well without much performance loss. The results also showed that planning further ahead negatively impacts the quality of planning

decisions, due to the increased difference between the predicted and realized state of the system. For this reason, we propose a hybrid method, in which 60% of appointments are fixed twelve weeks in advance using a static roster, and the remaining 40% of appointments are allocated six weeks ahead using the ILP or decision rule. This allows many appointments to be booked far in advance, while improving the quality of planning decisions due to the more accurate state prediction. This method shows an improvement in the number of patients that are treated within their urgency level deadline, compared to the static roster or using the (I)LP to schedule all appointments twelve weeks in advance.

Some more practical recommendations can be made; further details on these can be found in Section 11.2.1. The first is that a lot of data must be collected in order to implement the solution methods, especially logistic path data which is used in the simulation and for determining transition probabilities. We noticed that some data regarding OR appointments were missing here, and that urgency levels of repeat consultations were not recorded. Furthermore, we want to stress again that the cost and reward parameters must be chosen carefully. Finally, we recommend investigating whether the (I)LP can be expanded from one surgeon to model a unit or even the entire orthopaedic department.

List of Abbreviations

Abbreviation	English name	Dutch name
ICU	Intensive care unit	Intensive care (IC)
SMK	Sint Maartenskliniek	Sint Maartenskliniek
OD	Outpatient department	Polikliniek
OR	Operating room	Operatiekamer
FC	First consultation	Nieuwe klacht (NK)
RC	Repeat consultation	Vervolg consult (VC)
DC	Discharge consultation	Ontslag consult (OC)
GP	General practitioner	Huisarts
MDP	Markov decision process	
(R-)LSTD	(Recursive) least squares temporal differencing	
LSPI	Least-squares policy iteration	
ADP	Approximate dynamic program(ming)	
(M)ILP	(Mixed) integer linear program	
EVI	Exact value iteration	

Contents

1 Introduction

The COVID-19 pandemic has shown us in an unprecedented way how critical healthcare capacity can be to the well-being of society. The shortage of ICU beds and hospital staff resulted in delays in elective care and surgeries in the Netherlands [1], and in countries such as India, hospitals were even forced to refuse patients [2]. Even in non-pandemic times, the Dutch health care system has been facing increasing pressure due to an ageing population and shortage of nursing and hospital staff [3]. This can lead to longer waiting times, lower quality of care, higher costs and overworked staff. One way to avoid such capacity shortages is to invest more in human and material resources. Another way is to make more efficient use of the resources already available by optimizing healthcare processes, schedules and resource allocation.

One such process that can be optimized, is that of scheduling surgeons treating elective patients. Most patients visiting a hospital for elective care first go through one or more consultations before undergoing surgery. In these consultations, the surgeon may want to determine the nature of the complaint, or whether the patient can undergo surgery, and which type of surgery is required. Hospitals often discern between different types of consultations, such as patients visiting the hospital for the first time, pre- and post-operative consultations, or follow-up consultations. One can imagine that, for example, scheduling many pre-operative consultations in one week could lead to higher waiting times for surgery in the next week, as many patients will be ready for surgery at the same time. It is desirable to schedule these consultations in such a way that patient waiting times remain stable and as many patients as possible receive treatment. This thesis researches various methods in the field of mathematical optimization which can be applied to planning consultation types in order to achieve this goal.

In Section 1.1 of this introductory chapter, we provide some context on the organizations featured in this thesis. We expand on the problem at hand in Section 1.2. The scope and research question are defined in Section 1.3. We discuss the contribution offered by this thesis to the field of healthcare capacity management in Section 1.4. Finally, Section 1.5 describes the structure of the remainder of this thesis.

1.1. Context

The problem, and solutions, explored in this thesis are relevant to health care capacity management in all hospitals, and can even be applied to capacity management problems in other domains. In order to evaluate the performance of solutions on real-world data, this thesis features a case study of healthcare planning at the Sint Maartenskliniek, performed under supervision of Rhythm.

1.1.1. The Sint Maartenskliniek

The Sint Maartenskliniek (SMK) is a hospital devoted to the treatment of medical conditions related to posture and movement [4]. Its main focus areas are orthopaedics, rheumatology, rehabilitation and pain treatment. Since it is a highly specialised hospital, it is known for its treatment of highly complex

conditions, and treats many patients that are unable to find treatment elsewhere. This thesis focuses solely on the orthopaedic department.

SMK has seven locations, three of which (Nijmegen, Boxmeer, and Woerden) perform orthopaedic care. Nijmegen is the largest location, whereas Woerden only has an outpatient department and so does not carry out surgeries. All three locations fall under one orthopaedic department and the surgeons employed there often work at multiple locations throughout the week. The orthopaedic department is subdivided into seven units: hip and upper leg, knee and lower leg, foot and ankle, back and neck, shoulder and arm, prosthetic joint infections, and paediatric orthopaedics. In 2019, it employed 33.1 FTE, who treated 32,027 patients and performed 8,118 surgeries throughout the year, producing a revenue of €86,920,000 [5]. This makes their orthopaedic department the largest one in the Netherlands [4].

1.1.2. Rhythm

The capacity planning department of SMK is supported by Rhythm, a company specialised in healthcare operations management. Rhythm is an initiative of the research group CHOIR (Centre for Healthcare Operations Improvement and Research) at the University of Twente, and the company ORTEC. They collaborate with various healthcare institutions to help them optimize their care processes. This is done by giving advice based on quantitative analyses, providing process support, and developing and implementing software [6].

1.2. Problem description

Patients of the orthopaedic department of SMK follow a treatment pathway consisting of consultations in the outpatient department, diagnostic tests and, if necessary, surgery. SMK discerns between three general types of consultations: first consultations (FC), repeat consultations (RC), and discharge consultations (DC). As the name suggests, a patient always starts their treatment with a first consultation, which may be followed by some repeat consultations. A discharge consultation occurs a few weeks after surgery in order to check if the surgery has had the desired effect and determine if any further treatment is needed, or if the patient can be discharged from the treatment process. The planning department of SMK determines how much time a surgeon should spend each week in the outpatient department (OD) on consultations, and in the operating room (OR) on surgeries. A surgeon's OD time is split into a predetermined number of FC, RC and DC consultations.

Since time is a limited resource, these planning decisions restrict the number of patients of each type that can be treated every week. Patients who cannot be treated must wait at least another week for their appointment. When waiting lists are long, waiting times will increase; this not only negatively impacts patient satisfaction, but can be dangerous to patients suffering from severe medical conditions. On the other hand, short waiting lists can lead to unused capacity when there are less patients waiting for treatment than there are appointment slots to be filled. Unused capacity is expensive and leads to higher healthcare costs, and it implies that fewer patients receive treatment than may have been possible, if better planning decisions had been made. Many hospitals employ similar treatment pathways and apply a similar healthcare planning structure, so the problem described here can be generalized throughout the healthcare sector.

It is important to understand the impact of planning decisions on waiting lists. This can lead to lower waiting times and more efficient use of available resource capacity. Better yet, models and tools can be developed to support the capacity planning department in their decision-making in order to reach these goals. In this thesis, we focus on a specific planning intervention with regards to this objective.

1.3. Scope

This thesis explores how healthcare planning can be used to control waiting lists with the objective to reduce waiting times and make efficient use of resources. Previously, research has been done on determining the required amount of OD and OR time per week for surgeons at SMK in order to control

OR waiting lists [7, 8]. This research is discussed in Section 5.1. Another factor influencing OD and OR waiting lists is the distribution of consultation types planned within OD sessions. These sessions are split into a number of timeslots. Each timeslot is allocated either a first, repeat or discharge consultation. Determining the number of each type of consultation to be scheduled, is referred to in this thesis as the *timeslot allocation problem*. Our aim is to model this problem, and find and implement appropriate solution methods.

Currently, when planners at SMK notice that a surgeon has a very long OR waiting list, they might schedule less first consultations for that surgeon, hoping that this will result in less OR appointments. However, if these are replaced by repeat consultations, this can actually lengthen the OR waiting list in the short term, as repeat consultations lead to OR appointments much sooner than first consultations do. In the long term, both the RC and OR waiting lists will dry up with a lower influx of new patients. Ignoring such effects when scheduling patients can cause large fluctuations in the lengths of waiting lists, negatively impacting waiting times for patients, and the number of patients that can be treated.

This thesis aims to identify and exploit the effects of adjusting the allocation of timeslots on waiting times and use of resource capacity. Note that there is somewhat of a trade-off between these two goals when it concerns the number of first consultations to schedule. Admitting fewer new patients allows for more time to treat patients currently on the waiting lists, reducing their waiting time, but can lead to wasted appointment slots and so less patients will actually receive the treatment they need. The variability and uncertainty resulting from this intervention further complicates the problem. In this thesis, we aim to answer the research question:

How can timeslot allocation of consultation types be used to control waiting times of patients and use of resource capacity?

1.4. Contribution

A lot of relevant research has been done so far in the field of healthcare capacity management, some of which is discussed and used throughout this thesis. We contribute to this field in a number of ways. First, we build on a model which can be used for timeslot allocation, presented by Hulshof et al. [9, 10], by introducing the concept of urgency levels and using a different contribution function. The authors use an integer linear program (ILP) in [9] and approximate dynamic programming in [10] to find solutions to the problem. However, these methods are never compared with each other, or implemented on real-world data. In this thesis, we formulate a similar (I)LP, and investigate a form of approximate dynamic programming suitable for infinite-time-horizon problems: least-squares policy iteration (LSPI). Four decision rules are also proposed. These methods are implemented on data from SMK and their performance is compared with the current planning method used by the hospital.

Secondly, we implement the concept of planning ahead. Since schedules must usually be generated weeks in advance, perfect information about the future state of the system for which we wish to make a planning decision is unavailable. In this thesis, we develop a method to make a prediction of that future state. Furthermore, we compare the performance of solution methods studied as this planning horizon is increased.

Finally, to the best of our knowledge, the variant of LSPI used in this thesis has not been used previously in literature. An off-policy variant using an approximate Q-function is introduced in [11], and an on-policy variant using an approximate value function is presented in [12]. We introduce and implement an off-policy variant using an approximate value function. Moreover, rather than using a post-decision state within the algorithm as is done in [10, 12], we argue why an expectation of the next state can and should be used. We also extensively investigate the influence of adaptations to hyperparameters and to the algorithm itself on convergence and performance on the timeslot allocation problem.

1.5. Thesis outline

Following this introductory chapter, Chapter 2 provides a situational analysis of healthcare planning at the Sint Maartenskliniek, by describing in more detail the patient pathway, as well as Dutch and self-imposed norms on waiting times. We move on to discussing a general framework for healthcare planning, and how this is applied at SMK, in Chapter 3. The mathematical background which is essential to understanding the rest of this report is given in Chapter 4. Chapter 5 provides an overview of related research in the field of healthcare planning, including research previously conducted at SMK and a model on which our research is based. Our model for the timeslot allocation problem is presented in Chapter 6, and the solution methods which are used to solve the problem are introduced in Chapter 7. These solution methods are applied to a small test problem, a large test problem and finally to the actual data provided by SMK. The data used for each of these problems are described in Chapter 8. We discuss the results of the small and large test problems in Chapter 9, while the results of the case study at SMK are discussed in Chapter 10. Finally, our conclusions and recommendations are presented in Chapter 11.

2

Situational analysis

This chapter provides an analysis of the current situation of healthcare planning for the orthopaedic department of the Sint Maartenskliniek. In Section 2.1, we describe the typical treatment pathway followed by patients, the deviations that have been identified from this typical pathway, and the difficulties that these cause for the planning process. We then analyse how well SMK performs regarding national norms on waiting times in Section 2.2. This provides a benchmark to which our own results can be compared later on.

2.1. The patient pathway

In its simplest form, the patient pathway at the orthopaedic department of SMK can be described as follows. A patient, having been referred to SMK by their general practitioner (GP) or by another hospital, starts off with a first consultation (FC) with a surgeon at the relevant unit. This is often followed by multiple repeat consultations (RC). These consultations are often paired with appointments at the radiology department for diagnostics, such as MRI, X-ray or CT scans. Based on this, the surgeon may decide that the patient should move on to surgery (OR), in which case an OR-ticket is written, or they terminate the treatment. Prior to a surgery, a patient must undergo a pre-operative screening. Patients who have had surgery should return after some time, usually six weeks, for a post-surgery discharge consultation (DC). If more consultations are required after this, these are scheduled as RCs. The process is visualized in Figure 2.1.

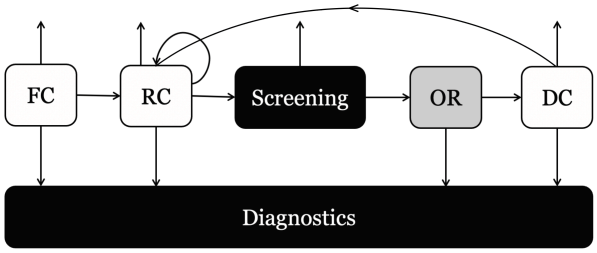

Figure 2.1: Patient pathway for orthopaedic care at SMK

The appointments indicated in white (FC, RC, and DC) all take place in the outpatient department (OD), and are conducted by the same surgeon who performs the surgery. Diagnostics are done by the radiology department, and screening is done by anaesthesiologists and assistant physicians. These stages are indicated in black as they are left out of our analysis, as our scope is restricted to the schedules of surgeons. Approximately 75% of all patients leave the process at the FC, RC or screening stage, for instance if it is decided that surgery is not necessary, they should be treated elsewhere, or if they are not fit for surgery.

Unfortunately, the pathway is almost never this simple. Through investigation of patient pathway data and interviews with employees of the capacity management department at SMK, many exceptions to this general model can be identified, such as:

1. The patient transfers to a different surgeon or unit in the orthopaedic department, sometimes shortly before undergoing surgery. In this case, the patient does not start their treatment with the new surgeon with a first consultation.
2. Often, a surgeon wants an extra check-up on a patient months or one year after their surgery; these appointments are planned as repeat consultations. The urgency of such appointments is usually much lower than regular repeat consultations, as they do not contribute to waiting time before a surgery.
3. Some surgeons schedule a repeat consultation between surgery and the discharge consultation if they require multiple short-term post-surgery checkups. Sometimes, no discharge consultation is scheduled at all, and all post-surgery appointments are repeat consultations. It is unclear if this is done on purpose, or if this is a mistake in data registration.
4. A patient may require multiple surgeries.
5. Patients may follow a pathway at multiple units simultaneously, for example if they have complaints in their shoulder and their hip. This does not necessarily complicate the pathway, as these tracks are generally kept separate, but this can occasionally cause errors in data registration.
6. An emergency situation may arise, leading to a patient being treated by the emergency department rather than their regular surgeon.
7. Data about the patient pathway may be recorded incorrectly.

These exceptions make it difficult to predict how long the waiting list of a surgeon will be in the future.

Another difficulty in the planning process is that typical patient pathways differ between surgeons and between the 7 healthcare units of SMK (hip and upper leg, knee and lower leg, foot and ankle, back and neck, shoulder and arm, prosthetic joint infections, and paediatric orthopaedics). A less experienced surgeon is more likely to treat patients requiring simple treatments, such as a standard hip replacement. Very experienced surgeons often have patients transferred to them shortly before surgery, if it turns out that the surgery required is very complex. Some units, such as hip and upper leg, are also generally easier than others, such as back and upper neck. In the easier case, patients will have fewer consultations before and after their surgery, shorter surgeries, and a higher probability of undergoing surgery. Furthermore, many surgeons work for multiple units, in which case the mixture of different patient types further complicates the prediction and planning process.

Finally, since SMK is a specialized hospital, they also treat many patients who cannot be treated at any other hospital, referred to as “special” patients. These are prioritized above regular patients as they often have more urgent medical conditions, and it is easier for regular patients to switch to a different hospital if they have to wait for a long time to be admitted. As a result, the pathways, urgency, and waiting times of such patients differ from those of regular patients.

2.2. The Treeknorm and urgency levels

The *Treeknorm* dictates the maximum acceptable access time within which a patient must be able to receive care in the Netherlands, as agreed upon by healthcare providers and insurance companies in [13]. This provides an important benchmark as to whether a hospital is able to provide care on time. When the Treeknorm is exceeded, patients are entitled to waiting list mediation through their health insurer.

External access time (EAT) is defined as the time between the moment a patient is referred to the hospital, and their first appointment at the hospital. The Treeknorm for external access times is that

this time should be within 3 weeks for 80% of patients, with a maximum of four weeks.

Internal access time (IAT) is the amount of time between a formal request for surgery, and the surgery itself. SMK discerns between five urgency levels, each with their own time limit:

- Within 24 hours (acute). These appointments are not within the scope of this thesis as they are not planned in advance.
- Within two weeks (urgent).
- Within one month.
- Within two months, with the exception of special patients, who should be treated within seven weeks.
- Within three months (elective), again with the exception of special patients, who should be treated within seven weeks.

Again, the norm is to meet these limits for 80% of all surgeries. It is especially important to adhere to these limits for patients in the first two categories, as they can face dangerous medical consequences if they are not treated on time, while patients in the other categories are more likely to only experience some discomfort if they have to wait for a longer period of time.

In addition to these norms, surgeons at SMK also assign a planning period to repeat and discharge consultations, indicating in how much time they want a patient to return for their next consultation. For discharge consultations, this is almost always six weeks after surgery. For repeat consultations, the time period can differ from three weeks to six months. Although there is no official obligation to stick to these deadlines, the timing of these appointments does affect how quickly patients can go through their treatment program and move on to surgery.

The percentage of patients whose external or internal access time is within the norm, and the average access times (relative to the norm), are good performance indicators for measuring the quality of the planning decisions made. Here, we present these statistics for the year 2019 (2020 and 2021 are ignored due to COVID-19), specifically for the unit that is investigasted in the case study. This serves as a benchmark to which our solution methods can be compared.

The three-week limit for EAT is met only [redacted] of the time, constituting a large difference with the norm of 80%. The average EAT is [redacted] days. The limits for IAT are met, on average, for [redacted] of all surgeries, which is better than the EAT but still far away from the norm. Table 2.1 shows the average IAT, and the percentage of surgeries performed within the time limit, per urgency level.

Table 2.1: Internal access times for one unit at SMK in 2019

Urgency level	Number of patients	Average IAT (days)	Surgeries within time limit
1 day (acute)	[redacted]	[redacted]	[redacted]
14 days (urgent)	[redacted]	[redacted]	[redacted]
30 days	[redacted]	[redacted]	[redacted]
60 days	[redacted]	[redacted]	[redacted]
90 days (elective)	[redacted]	[redacted]	[redacted]

The data shows that the deadlines for acute and urgent patients are met most of the time. However, the majority of patients require elective surgery, and the 90-day deadline is only met [redacted] of the time. It is clear that there is room for improvement.

3

Healthcare planning

Healthcare planning is a broad field, concerned with everything from strategic, multiple-year plans to booking appointments and emergency scheduling. It is important to understand how the specific intervention studied in this thesis fits within this field. This provides intuition about which information can be used as input for a model, the stakeholders involved, and what the impact of planning decisions are on the rest of the planning process. We first introduce a general healthcare planning framework in Section 3.1. Then, Section 3.2 describes how this framework is applied at SMK, and how the timeslot allocation intervention fits within this framework.

3.1. Healthcare planning framework

There are many different frameworks describing healthcare planning. This thesis is based on the framework proposed by Hans et al. [14] as this is most similar to the framework used at SMK. They discern strategic, tactical, and offline/online operational levels of control, for medical, resource capacity, material, and financial planning managerial areas.

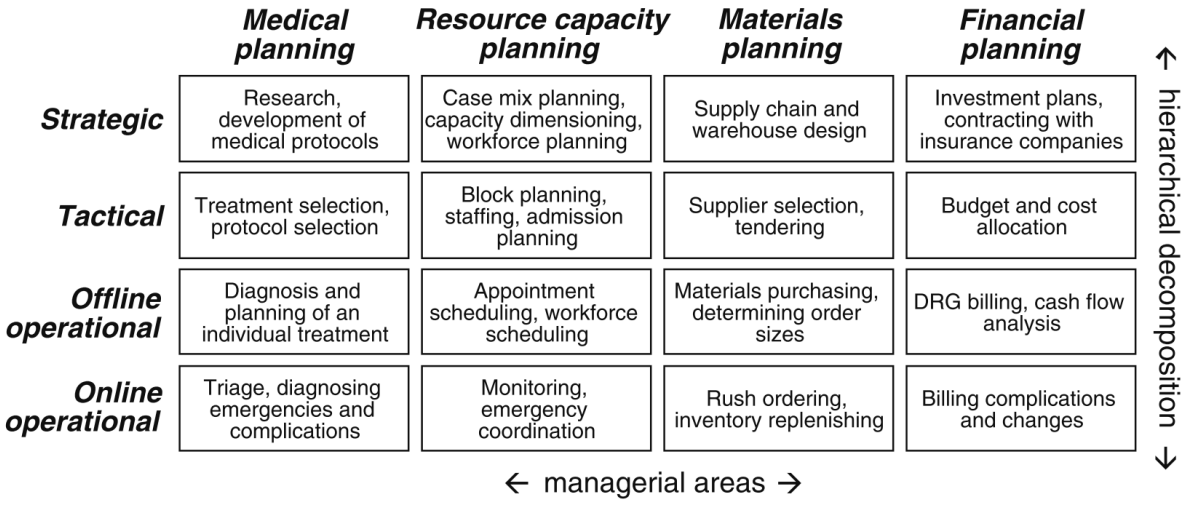

Figure 3.1: Healthcare planning framework proposed by [14].

This thesis is concerned with the resource capacity branch: the planning, scheduling, and control of resources such as equipment, facilities, and staff. At the strategic level, long-term goals are set, for example, determining the yearly estimates for the number of surgeries to be performed by each surgeon. These goals are often based on aggregated historical data, forecasts, and contracts with health

insurance companies. Operational planning involves day-to-day decisions, such as booking patient appointments.

Tactical planning finds itself in between these two levels, being more precise than strategic yet broader than operational planning. The main objectives of tactical planning are "*to achieve equitable access and treatment duration for patient groups, to serve the strategically agreed target number of patients (i.e., production targets or quota), to maximize resource utilization and to balance workload*" [9]. This implies outlining the division of labour and resources in accordance with strategic goals, providing a blueprint based on which operational planning can be done.

3.2. Planning at SMK

The intervention studied in this thesis is done at the end of the tactical planning phase. It is also important to understand how strategic and operational planning are done at SMK. The former provides the targets tactical planning should strive towards, while the latter informs us on the structure in which tactical plans should be made. Most of the information provided in this section has been obtained from interviews with members of the capacity planning department at SMK.

3.2.1. Strategic planning

Every year, the hospital makes an annual budget outlining (among other things) how many surgeries each unit within SMK should perform. This is done in collaboration with health insurance companies and is based on historical data, forecasts for the upcoming year, and relevant changes to the hospital's policy and resources. Based on this budget and the available capacity, planners calculate how many surgeries each surgeon of each unit should perform per year in order to meet the unit targets. They also calculate how many days a surgeon should spend in the operating room per week (on average) in order to reach their personal target. A target waiting list length is also calculated, which indicates how long the waiting list for OR appointments of each surgeon should ideally be in order to meet their personal target while maintaining waiting times to an acceptable level.

3.2.2. Tactical planning

Tactical planning at SMK involves making an activity plan, which is then used to generate an activity roster. Input for this phase consists of the strategic targets, current workload of the surgeons, available resources and requested absences of the staff. The output is a roster indicating which activities (OD, OR, administrative work, research etc.) should be done during which dayparts (half days) over a two-week period. This phase is done twelve weeks in advance, so that surgeons know which hours they are required to work, and patients can be notified of their appointments in time. The tactical planning timeline is visualized in Figure 3.2.

First, an *activity plan* is generated for each surgeon, which determines the number of dayparts a surgeon should spend on each type of activity over a two-week period. Most surgeons have a fixed number of dayparts that should go towards administration or research. This usually leaves around 3 to 3.5 days per week which can be dedicated to OR and OD appointments. The assigned number of OD and OR dayparts is obviously the largest restriction on the number of patients a surgeon can treat per week. Therefore, it often does not suffice to blindly use the average number of OR days per week determined in the strategic planning phase. The planners must also take the expected workload of the surgeon into account. For example, a surgeon with a very long OR waiting list should probably be given relatively more OR time in order to reduce it.

Next, the *activity roster* is generated using optimization software. This determines which activities are done during which dayparts (for example, administration on Monday morning and an OD session on Monday afternoon). The roster takes available locations and (downstream) resources into account. Surgeries within one unit often require the same type of OR resources, so these must be spread out throughout the week. There is also a limited capacity of beds available for patients to recover after their surgery.

The roster also determines how much supporting staff is delegated to each surgeon during their OD sessions. Supporting staff include assistant physicians, nurse practitioners, interns and residents. An assistant can hold consultations in parallel to the surgeon, so more patients can be seen in the OD session. However, assistants still require some guidance from the surgeon and so the number of extra patients that can be booked depends on the experience level of the assistant. Zero, one or two assistants can be assigned to a surgeon during an OD daypart, referred to as *uno, duo* and *trio* sessions, respectively.

Finally, the roster is approved at the tactical planning meeting with the heads of the units, and the capacity planning department can move on to planning the following two weeks.

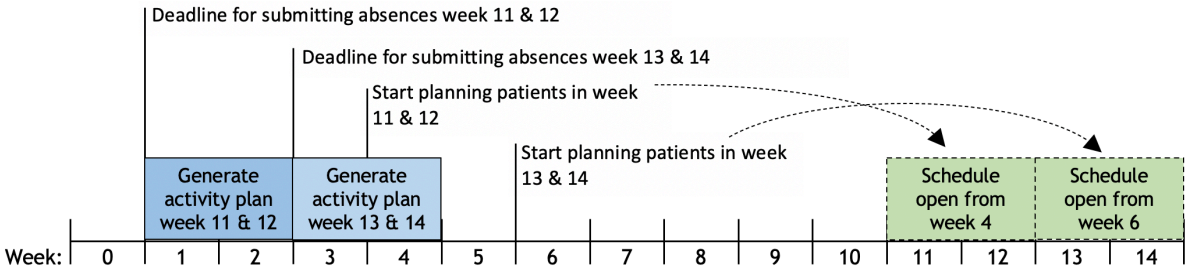

Figure 3.2: Scheduling timeline at SMK [15]

3.2.3. Operational planning

After the activity roster has been made, appointments can be booked for patients. An OD daypart is split into timeslots, usually of 15 minutes, in which appointments can be booked. If the daypart is a duo or trio session, the number of timeslots in the daypart can be increased accordingly. A fixed framework indicating which timeslots should be allocated to FC, RC, and DC appointments is applied to the OD dayparts. These slots can then be filled by patients. Patients are booked in timeslots according to their availability, access time, and urgency. Surgeries are somewhat more difficult to plan as the duration of a surgery is variable and depends on the type of surgery required.

The process described here falls under offline operational planning. Online operational planning consists of coordinating emergency appointments and surgeries, and is not in the scope of this thesis.

3.2.4. Timeslot allocation within the planning framework

The intervention of changing the number of first and repeat consultations falls in between the tactical and operational planning phases. The required input is the number of available OD timeslots, the amount of time a surgeon will spend in the OR and the current workload of the surgeon. This information can be obtained from the activity roster, which is generated in the tactical planning phase. The output is the number of FC, RC and DC appointments that should be scheduled in the available time. This is used to make the appointment framework used in the operational planning phase.

4

Mathematical background

This chapter provides the mathematical background knowledge essential to understanding the solution methods discussed in this thesis, and in related research. We show later on that the timeslot allocation problem can be modelled as a Markov decision process. Section 4.1 is devoted to background on Markov decision processes, and how solutions can be found using exact or approximate value- or policy-iteration. Another solution method that is used in this thesis is linear programming. Mathematical background on this topic is presented in Section 4.2.

4.1. Markov decision processes

Tactical planning for health care can be described as a *sequential decision process under uncertainty*. Every two weeks, planners at SMK must make decisions regarding the type of appointments to schedule, taking into account future implications of these actions even though the outcomes of those actions are uncertain. More generally, a sequential decision problem consists of several decision problems at discrete time steps, in which the outcome at each step influences the future decision problems. A decision problem under uncertainty arises when the consequences of decisions are not fully known or stochastic. A sequential decision process under uncertainty unifies these problem types. The (relevant features of the) environment at each time step make up the states of the problem. When costs or rewards can be assigned to (outcomes of) decisions, and the Markov property holds, which is that future states only depend on the current state and not on previous states, such a problem can be formulated mathematically as a Markov decision process [16].

A Markov decision process (MDP) can be characterized by the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, C)$ with set of states $\mathcal{S}$, set of actions $\mathcal{A}$, set of time steps $\mathcal{T}$, set of transition probabilities P, and a contribution (or cost) function C [17]. The states $\mathcal{S}$ define the (features of the) environment in which the problem takes place, such as the length of waiting lists or waiting times of patients. The set of actions $\mathcal{A}$ determines which actions the agent can perform to control the system at each moment in time, for example how many patients should be treated. These actions are performed within the finite set of time steps $\mathcal{T}$. When an action a is applied to a state s, the resulting state will be state s' with probability $P(s'|s,a)$. Finally, the contribution (cost) function C specifies some contribution (cost) for being in a certain state ($C : \mathcal{S} \to \mathbb{R}$), performing an action in a state ($C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$), or undergoing a certain transition between states ($C : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$).

Sometimes, a stochastic transition function $T(s,a,X)$ is defined rather than transition probabilities, where X represents exogenous information, which is (stochastic) information learned after a decision is made. The output of the transition function is the next state, s'. It can also be useful to define an approximate function $\overline{T}(s,a)$, which provides an approximation of the next state if stochastic information cannot be used. This function could provide an expectation of the next state, or alternatively a *post-decision state*, as proposed by Powell in [12]. This is the state that would result from the deterministically known outcomes of performing action a in state s, before (or without) observing exogenous

information X. For example, when a surgeon treats a patient we know (deterministically) that the patient will leave the waiting list they are currently waiting in; this is the post-decision transition. The exogenous (stochastic) information could be which type of appointment, if any, the patient should be scheduled for next; this is the full transition.

The objective of solving an MDP is to find a policy maximizing the expected contribution (or minimizing the expected cost) over time. A policy is a decision function $\pi : \mathcal{S} \to \mathcal{A}$ determining which action to take given a state. We move through the process in the following way: given an initial state s_0, the policy determines an optimal action $a_0 = \pi(s_0)$. This results in a new state s_1 with probability $P(s_1|s_0, a_0)$ and a contribution $c_0 = C(s_0, a_0)$ (in the case of the second type of contribution function). The process continues, providing a chain $\{s_0, a_0, c_0, s_1, a_1, c_1, \dots, s_T, a_T, c_T\}$. The expected contribution is given by $\mathbb{E}\left[\sum_{t\in\mathcal{T}} c_t\right]$. If the time horizon is infinite, or we wish to discount future contributions in some way (since predictions further into the future may be more unreliable), it is also possible to use the expected discounted contribution $\mathbb{E}\left[\sum_{t\in\mathcal{T}} \gamma^t c_t\right]$ where $\gamma \in [0, 1]$. Of course, setting $\gamma = 1$ in this case simply provides the full expected contribution.

The value of a state $s \in \mathcal{S}$ under policy π is the expected contribution gained by following π when starting in state s. This is given by the value function:

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t\in\mathcal{T}} \gamma^t C(s_t, \pi(s_t))|s_0 = s\right], \forall s \in \mathcal{S}.$$

Denote the optimal policy by π^* and its corresponding value function by V^*. By the principle of optimality, the optimal solution satisfies the Bellman equation [18]:

$$V^*(s) = \max_{a\in\mathcal{A}}\left(C(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s, a)V^*(s')\right), \forall s \in \mathcal{S}. \tag{4.1}$$

Another way to represent the value function, which is more popular in the computer science community for its use in reinforcement learning, is using a Q-function. A Q-function specifies the value $Q(s, a)$ of performing action a in state s. In that case, the Bellman optimality equation becomes:

$$Q^*(s, a) = C(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s, a) \cdot \max_{a'\in\mathcal{A}}(Q^*(s', a')), \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{4.2}$$

The value function can be found from the Q-function as $V(s) = \max_{a\in\mathcal{A}} Q(s, a)$.

4.1.1. Solving an MDP

Equation (4.1) can be solved using dynamic programming. However, in most practical cases (such as the problem under consideration in this thesis), this can be impractical (or even infeasible) due to the three 'curses of dimensionality' [12]:

1. The state space $\mathcal{S}$ may be too large to evaluate the value function for all states,
2. The action space $\mathcal{A}$ may be too large to find the optimal action for each state, or
3. The outcome space, given by all possible states s' resulting from taking action a in state s, may be too large to compute the expectation for future values.

When one (or multiple) of these apply, it may be necessary to develop approximation methods or use some form of aggregation in order to solve the problem in reasonable time.

Another issue is the assumption that everything about the system is known. In particular, the transition probabilities P may be unknown in practice. One possibility is to estimate these from data. Another approach is to simulate direct interaction with the environment, resulting in samples of state transitions and contributions, from which either the model or the optimal policy is learned. This is known as

model-free learning, whereas assuming all information is known is called model-based.

The problems described here have led to the development of many different algorithms for solving MDPs. These all fall under one of two strategies: value or policy iteration. In value iteration, the value function is estimated in each iteration, which in turn defines a policy. In policy iteration, a policy is defined in each iteration, and then the value function for that policy is calculated. This is used to evaluate and improve the policy, and so forth. Of the two, value iteration is most popular as it is the simplest to implement. Dynamic programming is an example of an exact model-based value iteration algorithm.

Exact value and policy iteration require looping over all states multiple times, and storing values in a lookup table. They also require knowledge of the transition probabilities. When this is not possible, we turn to approximate value or policy iteration [12]. These algorithms use linear or even non-linear parametrisations of the value function. In the linear case, such an approximation consists of two parts: F basis functions $\phi_1, \dots, \phi_F : \mathcal{S} \to \mathbb{R}$, and a parameter vector $\theta \in \mathbb{R}^F$. The approximate value function is given by:

$$\overline{V}(s) = \sum_{f=1}^{F} \theta_f \cdot \phi_f(s) = \theta^\top \phi(s).$$

The basis functions can be seen as an aggregation of the state space with F so-called features. The parameter vector provides a weight for each feature. The objective is to learn the parameter vector such that $\overline{V}(s)$ most closely resembles the optimal value function $V^*(s)$. Similarly, the field of approximate reinforcement learning attempts to learn the approximate Q-function:

$$\overline{Q}(s,a) = \sum_{f=1}^{F} \theta_f \cdot \phi_f(s,a) = \theta^\top \phi(s,a).$$

There are many ways to construct these basis functions and learn the parameter vector, of which an extensive overview is given by Busoniu et al. in [19]. For example, the parameter vector can be updated using linear gradient updates, but it is also possible to use regression trees or neural networks for this purpose. General basis functions include Gaussian radial functions, multilinear interpolation and crisp discretization. All of these methods fall under the umbrella of *approximate model-free value/policy iteration*. In the remainder of this section, we discuss two approximate policy iteration algorithms that use least-squares temporal differencing to update θ, and can be applied to infinite-horizon problems.

4.1.2. Least-squares policy iteration

An approximate policy can be expressed in terms of the approximate value function and approximate transition function [12], as

$$\overline{\pi}(s|\theta) = \arg\max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \overline{V}\left(\overline{T}(s,a)\right)\right).$$

Here, we maximize over the direct contribution resulting from performing action a in state s, and an approximation of future contributions. The approximate transition function $\overline{T}$ is used as an approximation of the next state, as we cannot enumerate over all possible next states. If a linear approximate value function is used, as is the case in this thesis, this becomes:

$$\overline{\pi}(s|\theta) = \arg\max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \theta^\top \phi\left(\overline{T}(s,a)\right)\right). \tag{4.3}$$

In case an approximate Q-function is used rather than an approximate value function, as in [11], the policy is defined as:

$$\overline{\pi}(s|\theta) = \arg\max_{a \in \mathcal{A}(s)} \left(\overline{Q}(s,a)\right) = \arg\max_{a \in \mathcal{A}(s)} \left(\theta^\top \phi\,(s,a)\right). \tag{4.4}$$

The goal is to learn the parameter vector θ which results in the closest approximation to the true optimal policy. We denote the true optimal policy by π^*, the best approximate policy by $\overline{\pi}^*$, and the parameter

vector resulting in this best approximation by θ^*.

Policy iteration algorithms cycle between two steps within a (simulated) environment: following the policy, and evaluating and updating the policy. These steps are often referred to as the "actor" and the "critic" [12]. The actor uses the current policy to select actions. This provides information about the consequences of those actions. After being allowed to act for some time, the critic evaluates the information gathered by the actor and uses this to calculate a new policy.

The actor can follow an *on* or *off-policy* trajectory. For the on-policy trajectory, a starting state s_0 is chosen. The actor uses the policy to choose an action a_0, observes the resulting contribution c_0, and transitions to the next state s_0' arising from performing action a_0 in state s_0 within the environment. The same process is repeated from state $s_1 = s_0'$, then $s_2 = s_1'$, and so forth, until a predetermined number (M) of states have been visited, or a steady state is reached. This is called on-policy because the policy determines which states are visited. In the off-policy case, the actor is given a dataset of M states $\{s_1, \dots, s_M\}$. For each state s_i, the actor uses the policy to choose an action a_i, and observes the resulting contribution c_i and following state s_i'. The order in which states are visited does not matter, and in this case, the policy only determines which action is chosen, not which states are visited.

There are two popular projection methods with which a policy in the form of Equation (4.3) can be evaluated and updated. These are the Bellman residual minimizing approximation [20, 21], and (recursive) least-squares temporal differencing (LSTD), first introduced by Bradtke and Barto in [22]. We focus on the latter as it has been shown to provide better experimental results [11].

After observing M state-action-contribution-state tuples (s_i, a_i, c_i, s_i'), the least-squares approximation to the θ that provides the closest approximation to the true value function for the current (not necessarily optimal) policy is given by [22]

$$\theta_M = A^{-1}b, \text{ where} \tag{4.5}$$

$$A = \frac{1}{M}\sum_{i=1}^{M} \phi(s_i)(\phi(s_i) - \gamma\phi(s_i'))^\top, \text{ and} \tag{4.6}$$

$$b = \frac{1}{M}\sum_{i=1}^{M} c_i\phi(s_i). \tag{4.7}$$

Note that the $\frac{1}{M}$ can be omitted as it is cancelled out when multiplying A^{-1} with b. If A is nonsingular and finite, which is the case if the basis functions are linearly independent, then $\theta_M \to \theta^*$ as $M \to \infty$ [23].

Note that Equation (4.5) involves computing the inverse of an $F \times F$ matrix after collecting a batch of M samples, which takes $\mathcal{O}(F^3)$ time. If F is large, we can improve on this using recursive least squares temporal differencing (R-LSTD), which takes $\mathcal{O}(F^2)$ time and $\mathcal{O}(F^2)$ space. R-LSTD computes A^{-1} recursively in every iteration using [22]

$$\alpha_m = \phi(s_m) - \gamma\phi(s_m'), \tag{4.8}$$

$$B_m = B_{m-1} - \frac{B_{m-1}\phi(s_m)\alpha_m^\top B_{m-1}}{1 + \alpha^\top B_{m-1}\phi(s_m)}, \tag{4.9}$$

$$b_m = b_{m-1} + c_m\phi(s_m). \tag{4.10}$$

B_0 is initialised as ϵI, where I is the $F \times F$ identity matrix, and $\epsilon > 0$ is a small constant; b_0 is initialised as $\mathbf{0} \in \mathbb{R}^F$. After M iterations, θ is estimated using

$$\theta_M = B_M b_M. \tag{4.11}$$

It is worth considering that this recursive computation can lead to a high build-up of computational errors, leading to a poor approximation. Furthermore, the non-recursive method requires far less matrix-vector multiplications. Therefore, the choice of whether or not to use the recursive method depends on

the relative sizes of F and M.

A policy iteration algorithm called least-squares policy iteration (LSPI) was first introduced in 2003 by Lagoudakis and Parr [11]. It is shown in Algorithm 1, and is referred to in this thesis as off-policy LSPI-Q. Note that the notation and structure shown here are different than in the original source, in order to remain consistent with the notation used in this document.

Algorithm 1: Off-policy LSPI-Q

1 Initialise $\theta_0 \in \mathbb{R}^F$
2 Generate a dataset of state-action-contribution-state tuples
$D = \{(s_m, a_m, c_m, s'_m) : m = 1, \dots, M\}$
3 Define the policy:

$$\pi(s|\theta) = \arg\max_{a \in \mathcal{A}(s)} \left(\theta^\top \phi(s,a)\right)$$

for $n = 1, \dots, N$ **do**
4 $b = \mathbf{0} \in \mathbb{R}^F$
5 $B = \epsilon I$, using a small constant ϵ and $F \times F$ identity matrix I
6 **for** $(s, a, c, s') \in D$ **do**
7 Perform the R-LSTD update:

$$\alpha = \phi(s,a) - \gamma\phi(s', \pi(s'|\theta_{n-1}))$$
$$B = B - \frac{B\phi(s)\alpha^\top B}{1 + \alpha^\top B\phi(s)}$$
$$b = b + c\phi(s)$$

8 **end**
9 $\theta_n = Bb$
10 **end**
11 **return** *the approximate policy* $\pi(s|\theta_N)$

The algorithm uses an off-policy actor and R-LSTD to update θ, and the policy is expressed using the approximate Q-function. Because of this, the dataset does not just contain states, but also samples of actions, and the contributions and following states resulting from those state-action pairs. An advantage of this is that it is not necessary to make or use a simulation, as a single dataset of real-world observations can be used.

In the algorithm, the policy is initialised with a parameter vector θ_0 (line 3), which is usually initialised as a random or all-ones vector (line 1). The algorithm then performs N iterations. In each iteration, for every tuple (s, a, c, s') in the dataset, the policy is used to determine the next action from state s'. This information is used within the R-LSTD update (line 7). After looping over the entire dataset, θ is updated (line 9), changing- and hopefully improving- the policy used in the next iteration.

In [12], Powell presents an on-policy version of the same algorithm where the policy is expressed using an approximate value function, which we refer to as on-policy LSPI-V. The algorithm is shown in Algorithm 2. Note that in this case, a dataset is not used. In every iteration, we start from some initial state, and use the policy and a simulation to generate the subsequent states and actions; as a result, the states visited in every iteration will be different. This algorithm can only be used when it is possible to simulate the system. The use of an approximate value function also makes it necessary to formulate an approximate transition function $\overline{T}$ (line 2).

Performance bounds have been proven which guarantee convergence of on-policy LSPI-V for finite M [24]. However, these require very strict conditions on the problem, and in practice there are examples where on-policy LSPI-V oscillates between two very poor policies. It is recommended to try both

on- and off-policy variants, and Q- and V-variants, in order to determine what works best for the problem at hand.

Algorithm 2: On-policy LSPI-V [12]

Initialise $\theta_0 = \mathbf{0} \in \mathbb{R}^F$
Define the policy:

$$\pi(s|\theta) = \arg\max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \theta^\top \phi\left(\overline{T}(s,a)\right)\right)$$

for $n = 1, \dots, N$ **do**
  Generate an initial state $s_0 \in \mathcal{S}$
  $b = \mathbf{0} \in \mathbb{R}^F$
  $B = \epsilon I$, using a small constant ϵ and $F \times F$ identity matrix I
  for $m = 1, \dots, M$ **do**
    Compute action $a_m = \pi(s_m|\theta_{n-1})$
    Obtain a sample of exogenous information x_m from the simulation
    Determine the next state $s_{m+1} = T(s_m, a_m, x_m)$
    Perform the R-LSTD update:

$$\alpha = \phi(s_m) - \gamma\phi(s_{m+1})$$
$$B = B - \frac{B\phi(s_m)\alpha^\top B}{1 + \alpha^\top B\phi(s_m)}$$
$$b = b + C(s_m, a_m)\phi(s_m)$$

  end
  $\theta_n = Bb$
end
return *the approximate policy* $\pi(s|\theta_N)$

Fortunately, LSPI contains few hyperparameters that should be tuned. The following choices are of importance when implementing the algorithm:

- The dataset of states D should be chosen carefully, as θ will fit best to the states in D. One could choose to generate random states, ranging across the entire state space, or use (close variations of) states which arise often in practice. This is an exploration-exploitation trade-off. Increasing M is likely to lead to a better approximation of θ, as more data is seen, but also increases the running time.
- The choice of basis functions is very important for the performance of the final policy. It is recommended to test different basis functions on a small test problem in order to determine which lead to the best results. This could be done by finding the exact solution, using linear regression to determine the best θ^* for each basis function compared to the exact solution, and then determining which basis functions lead to the best approximation of the exact solution. The same approach is carried out in [9]. Alternatively, LSPI can be tested using different basis functions on a simulation in order to determine which leads to the best performing policy.
- The limit on the number of iterations in the outer loop, N, can be replaced by a convergence criterion such as $||\theta_n - \theta_{n-1}|| < \delta$.

4.1.2.1. Applications of LSPI

Least-squares policy iteration has already been used for a variety of applications, such as path planning for mobile robots [25], dialog management [26] and self-learning cruise control [27]. The articles cite advantages such as LSPI being applicable to large and/or continuous state spaces, supporting both on- and off-policy learning, and being able to generalize well when the dataset is sampled randomly, as reasons for using the algorithm. Often, LSPI is combined with another algorithm, such as the A* algorithm

in [25], which handles the first part of a path planning process, or fast feature selection in [26], which efficiently selects the features to be used in LSPI.

An interesting application in the context of this thesis is the research of Schutz and Kolisch [28], who use simulation-based approximate dynamic programming (ADP) to allocate resource capacity to customers in the service industry. Customers can request use of some service during a booking period, at which point the service provider must decide whether to accept or reject the customer. This decision is based on the amount of capacity still available, and the expectation of which customer types (high or low priority) might still arrive at a later time. They assume that customer arrivals, service times and cancellations or no-shows are stochastic. An example of such a problem is allocating radiology appointments to patients. The problem is modelled as a continuous-time MDP. They show that simulation-based ADP, a value-iteration variant of LSPI, is suitable for solving the problem.

4.2. Linear programming

We show later that the timeslot allocation problem can also be formulated as a linear program (following some simplification), and solved accordingly. In this section, we provide some background on (integer) linear programs and how these can be solved.

A linear program is made up of three components: variables, an objective function expressed in those variables, and constraints on the variables [29]. The goal is to maximize or minimize the value of the objective function, such that the variables do not violate the constraints. The objective function and constraints are linear functions (hence the term *linear* programming). The variables can take on real values, unless otherwise specified.

In general, a linear program can be formulated as:

$$\max_x \{c^T x : Ax \leq b, x \geq 0\}, \tag{4.12}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and x is an n-dimensional variable vector [30]. A and b define the constraints on x, and c provides the weights in the objective function. If x is constrained to only integer values, the problem is called an integer linear program. Similarly, if x is constrained to $\{0, 1\}$, the problem is called a binary integer program.

We provide a small intuitive example. Suppose we want to decide how much to produce of two products, x_1 and x_2. We gain €5 per unit of x_1 and €3 per unit of x_2 produced. Our goal is to maximize the total profit. We have 10 units of a resource needed to produce the products, but x_1 requires twice as much of the resource than x_2. Furthermore, we can produce at most 3 units of x_2. We can only produce a whole number of each product. This problem can be formulated as the following integer linear program:

$$\max_x \quad 5x_1 + 3x_2 \tag{4.13a}$$

$$\text{s.t.} \quad 2x_1 + x_2 \leq 10, \tag{4.13b}$$

$$x_2 \leq 3, \tag{4.13c}$$

$$x_i \in \mathbb{Z}^+, \quad \forall i = 1, 2. \tag{4.13d}$$

We can quickly see that the optimal solution is $x_1 = 4, x_2 = 2$ with a total of €26 profit. Of course, not every linear program is so simple, and cannot be solved by hand.

A linear program can be solved in polynomial time. However, the most popular method to solve a linear program is the simplex algorithm, which can take exponential time in the worst case but usually takes polynomial time [31]. Until now, (binary) integer linear programs cannot be solved in polynomial time (unless P = NP). Many different solvers exist which can efficiently compute (optimal) solutions for mathematical optimization problems, among which (integer) linear programs. In this thesis, the solver Gurobi [32] is used in combination with the programming language Python.

5

Related research on healthcare planning

A lot of research has been done on healthcare planning, especially within the fields of operations research and optimization. Developing techniques to improve healthcare planning decisions not only saves time and money but, more importantly, improves quality of health of patients. In this chapter, we discuss some of this research, focussing on research related to the timeslot allocation problem. First, Section 5.1 presents research done previously at SMK on planning OD and OR sessions. Then, Section 5.2 discusses consequences of a similar planning intervention at a different Dutch hospital. Finally, Section 5.3 describes research done on the timeslot allocation problem, and two solution methods, on which our model is based.

5.1. OD and OR session allocation

Prior to this thesis, similar research was done by Hatting [7] and Tsai [8] under supervision of Rhythm on healthcare planning in the orthopaedic department of SMK. Their work was aimed at determining an optimal allocation of OD and OR sessions for orthopaedic surgeons at SMK, in order to minimize waiting times for patients requiring surgery, reduce unused OR time and meet the yearly production targets of each surgeon. This goal was expressed as reducing fluctuation around a target OR waiting list length. Both modelled the patient pathway at SMK as a Markov decision process, but applied different methods to solve the problem.

Tsai [8] developed a detailed model which included the diagnostics and screening process. Stochastic dynamic programming was applied to determine an optimal allocation, but this was impractical due to high computation times for calculating transition probabilities between states. Even after applying simplifying assumptions to the model, which significantly reduced the state space, the model remained computationally expensive. Tsai recommends ignoring the radiology department and mentions approximate dynamic programming as a possible solution method.

Hattingh [7] formulated a similar MDP model to Tsai, following the recommendation to ignore the radiology department. Test cases where the OD and OR hit-rates were modelled as either stochastic or deterministic were compared. Replacing the stochastic parameters by deterministic averages resulted in similar solutions for the test cases and significantly reduced the running time. This assumption allowed for the formulation of an integer linear program in order to optimize the model. The ILP was successfully able to determine the number of OD and OR sessions that should be scheduled for each surgeon in a two-week planning period.

These results could then be used as an input for the scheduling prototype application developed by Hattingh in the same thesis, which determines on which days the OD and OR sessions should be planned for each surgeon. The model takes into account the effects of the schedules of all surgeons on downstream resources such as the radiology department and recovery ward. OD appointments are often followed by radiology appointments, while surgery results in use of beds in the recovery ward. The

aim is to evenly distribute the expected utilization of these resources throughout the week. The model must take into account restrictions on resource capacity, such as time and available locations. Hattingh formulated an ILP which was implemented in AIMMS and was successfully able to generate schedules for all surgeons within a few minutes, reducing the fluctuation of use of downstream resources.

5.2. Effects of increasing capacity on access times

The Sint Maartenskliniek is not the only hospital suffering from variability caused by the ratio of new to returning patients. Roemeling et al. [33] studied the rheumatology department of a hospital in Groningen which wanted to reduce its external and internal access times. They increased their capacity by hiring a new resident physician and intern to the department, in order to be able to treat more new patients. At first, this significantly reduced the external access times from an average of 10 to 4 weeks. However, taking on many new patients without taking into account consequences earlier and later in the chain created an avalanche effect overwhelming the outpatient department.

GPs often select hospitals with the shortest external access times when referring patients. So, when external access times were reduced, there was a sharp increase in patient referrals to the hospital. Normally, this has the effect of balancing external access times between hospitals. In this case, the high inflow of new patients had a multiplier effect on the demand for repeat consultations, as each new patient required on average four RCs. These RCs absorbed all the added capacity, which eventually had an even stronger negative impact on the external access times than the increased number of referrals.

Now, although more patients could be treated due to the increased capacity, the promise of short external access times translated to long internal access times for those patients, and longer external access times for future patients. The authors stress the importance of awareness about the system, i.e., the referral behaviour of GPs and the multiplier effect of new patients to repeat consultations.

5.3. Patient admission planning

A very similar problem to the one under consideration in this thesis was investigated by Hulshof et al. in [9] and [10]. They describe a general model for tactical resource allocation and patient admission planning in health care, which is formulated as an MILP in [9] and as an MDP in [10]. The model fits our timeslot allocation problem very well and is therefore discussed in more detail here.

The model can be described as follows. The set of time-steps consists of a series of consecutive timeslots; $\mathcal{T} = \{1, 2, \dots, T\}$. There is a set of resources $\mathcal{R} = \{1, 2, \dots, R\}$, and patient queues $\mathcal{J} = \{1, 2, \dots, J\}$. Each queue $j \in \mathcal{J}$ requires some capacity $s_{j,r}$ from resource $r \in \mathcal{R}$. The resource capacity $\eta_{r,t}$ limits the number of patients that can use resource r at time t. For example, queues can be seen as different types of consultations or surgery, and resources could be time, locations or materials.

If a patient from queue $i \in \mathcal{J}$ is treated (i.e. they have been allocated their required resource in some time period), they can either leave the system, or transfer to another queue $j \in \mathcal{J}$ with a given probability $q_{i,j}$. Patients may also enter the system; the number of patients arriving in queue j at time t is given by $\lambda_{j,t}$. The set $\mathcal{U} = \{1, 2, ..., U\}$ represents the possible times that a patient can be waiting in a queue; this is upper bounded in order to ensure a finite state space. Patients can therefore be described by which queue they are waiting in during which time period, and how long they have been waiting for. The variable $S_{t,j,u}$ represents the number of patients in queue $j \in \mathcal{J}$ at time $t \in \mathcal{T}$ with waiting time $u \in \mathcal{U}$. Now, the state of the system at time t can be described by a vector $S_t = (S_{t,j,u})_{j \in \mathcal{J}, u \in \mathcal{U}}$. The decision variable is given by $x_{t,j,u}$: the number of patients from queue $j \in \mathcal{J}$ with waiting time $u \in \mathcal{U}$ who should be treated at time $t \in \mathcal{T}$.

5.3.1. Mixed integer linear programming

The first approach is to formulate the problem as a mixed integer linear program, which can be solved using the optimization software AIMMS. Since an MILP cannot solve stochastic problems, the transition probabilities are modelled as deterministic fractions q_{ij} of patients moving from one stage (or

queue) to another. The objective is "to achieve equitable access and treatment duration for patient groups and to serve the strategically agreed number of patients" [9]. In the objective function, this is approximated by minimising the weighted sum of patients waiting in each queue over all waiting times and time periods.

The authors apply an iterative scheme in order to determine the weights used in the objective function, so that the true performance targets set by the hospital are optimized. For relatively large instances (fifty queues, two resources, and six time periods), calculation time took on average four minutes with an integrality gap of 0.01%. Results show that the model divides resources more equitably over time and the number of patients served is closer to the strategically set target. The authors recommend implementing this model using a rolling-horizon approach, i.e., only applying decisions for earlier time periods while considering expected effects on later time periods. The model is relatively easy to adapt with, for example, more constraints or a different objective function.

5.3.2. Approximate dynamic programming

In [10], Hulshof et al. formulate the problem as an MDP. The objective is to generate a schedule which provides equitable access to treatment for patients and meets the strategic goals of the hospital. The cost function at time t is defined as

$$C_t(S_t, x_t) = \sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{J}} c_{j,u}(S_{t,j,u} - x_{t,j,u})$$

This assigns some predetermined cost $c_{j,u}$ to the number of patients who are not treated at time t. We can attach to the set of waiting times $\mathcal{U}$ some notion of patient priority, which is enforced by assigning different costs to different values of u. The aim is to determine a policy minimizing the expected future costs. This can be done by solving the Bellman equation

$$V_t(S_t) = \min_{x \in \mathcal{X}_t} (C_t(S_t, x_t) + \mathbb{E}\left[V_{t+1}(S_{t+1} | S_t, x_t, W_t)\right])$$

where $\mathcal{X}_t$ is the set of feasible decisions and W_t is the stochastic vector containing all new information (of patient arrivals and queue transitions) at time t.

Computing the exact solution using dynamic programming is only possible for very small instances, as the state, decision and outcome spaces grow exponentially in size (the three curses of dimensionality). Therefore, approximate dynamic programming (ADP) is applied, which is a form of approximate model-free on-policy value iteration. The algorithm overcomes intractability by using a post-decision state and a linear parametrisation of the value function using basis functions to approximate future costs. An ILP is used to efficiently find an optimal action given the current approximate value function and problem constraints. Various basis functions are proposed and compared using regression analysis. Recursive least squares is used to update the parameter vector used for the value function.

The authors use small test instances to show that the ADP algorithm provides a close approximation to the outcome of exact dynamic programming in reasonable time. For large instances (forty queues, four resources, eight time periods), the algorithm takes approximately one hour to converge to a parameter vector. The resulting policy performs much better on test instances than two proposed greedy heuristics. The authors conclude that ADP is an appropriate technique for tactical healthcare planning.

6

Modelling timeslot allocation

This chapter describes a model which can be used for timeslot allocation at the Sint Maartenskliniek, in order to minimize waiting times and maximize patient treatment. First, necessary assumptions are stated. Next, we describe the sets, parameters and variables defining the model. An objective function is proposed for the goals stated previously. Finally, solution methods are discussed which can be used to find an optimal solution to the problem.

We aim to model the effects of allocating timeslots to different appointment types on the waiting lists of a surgeon, based on the planning framework described in Chapter 3 and patient pathway described in Section 2.1. The input is the number of OD and OR timeslots available in the planning time period, which can be determined from the activity roster generated in the tactical planning phase. Relevant parameters for the surgeon and unit, and the current state of the waiting lists, should be given as well. The model should output how many OD timeslots should be allocated to which consultation types per time period.

Since treating patients not only has an effect on the current waiting lists but also on the future, as patients move through their treatment process, future consequences of current decisions must be taken into account. Additionally, since no patient is the same, these consequences are uncertain and we are faced with a sequential decision problem under uncertainty. For this reason, the problem is modelled as a Markov decision process (MDP). We refer to Section 4.1 for mathematical background on Markov decision processes.

6.1. Assumptions

The following assumptions have been made in order to simplify the model:

1. The total number of available OD and OR timeslots are given for each time period. There is sufficient capacity of other resources (locations, materials) to utilize the full timeslot capacity.

2. The time capacity required for each OD and OR appointment is fixed. For OD appointments this is a reasonable assumption as these are scheduled in 15-minute timeslots. OD appointments can take up multiple timeslots. This is not necessarily the case for OR appointments, as the time required depends on the complexity of the surgery. Rounded averages for the specific surgeon are used.

3. The diagnostics and screening stages (radiology department) can be ignored.

4. A surgeon is not affected by, or dependent on, activities of other surgeons.

5. Patient appointments are booked into appointment slots in the most efficient way. We elaborate on this assumption in Section 6.4.

6.2. Base model

Following the general model provided by Hulshof et al. [10] discussed in Section 5.3, the patient pathway for orthopaedic surgeons at SMK can be modelled as follows:

- Since planning is done at the tactical level, each time period represents two weeks. An infinite time horizon should be considered, in order to take into account all future consequences of actions. Unless stated otherwise, we set $\mathcal{T} = \mathbb{N}$.
- The patient queues are $\mathcal{J} = \{\text{FC, RC, OR, DC}\}$, representing the first consultation, repeat consultations, surgery and discharge consultation, respectively.
- The number of time periods that a patient has been waiting in a queue is indicated by the value $w \in \mathbb{N}$. For some solution methods, it may be necessary to upper bound the waiting time with some value W.
- Hulshof's model is extended with urgency levels $\mathcal{U}_j \subset \mathbb{N}$ for each queue $j \in \mathcal{J}$. These urgency levels represent the number of time periods for which it is acceptable for a patient to wait in a queue; after this time, the patient is considered to be treated too late. For example, typical urgency levels for the OR at SMK are two weeks, one month, two months or three months, so $\mathcal{U}_{OR} = \{1, 2, 4, 6\}$. Essentially, this is equivalent to replacing a queue $j \in \mathcal{J}$ by $|\mathcal{U}_j|$ queues corresponding to the different urgency levels, but using urgency levels makes notation somewhat easier.
- We consider a set of resources $\mathcal{R} = \{\text{OD, OR}\}$ representing timeslots in the outpatient department and operation room, respectively. The resource capacity $\eta_{r,t}$ indicates how many appointments can be scheduled for resource $r \in \mathcal{R}$ at time $t \in \mathcal{T}$. These capacities may vary depending on the previously determined activity plan. For example, if a surgeon receives OD assistance from one or two assistant physicians, $\eta_{OD,t}$ should be increased accordingly.
- The variable $\zeta_{j,r}$ indicates how much capacity of each resource $r \in \mathcal{R}$ is required by each patient in queue $j \in \mathcal{J}$.
- The number of new patients arriving from outside the system to queue $j \in \mathcal{J}$ with priority $u \in \mathcal{U}_j$ at time $t \in \mathcal{T}$ is given by $\lambda_{j_u,t}$.
- A patient in queue $i \in \mathcal{J}$ with urgency level $v \in \mathcal{U}_i$ transfers to queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$ with probability q_{i_v,j_u} when they are treated. Alternatively, it is possible to treat transition probabilities as unknown and simulate patients from data instead.
- The sub-state $s_{j_u,w,t}$ represents the number of people waiting in queue $j \in \mathcal{J}$ with urgency $u \in \mathcal{U}_j$ for $w \in \mathbb{N}$ time periods at time $t \in \mathcal{T}$. The state s_t is a vector $(s_{j_u,w,t})_{j\in\mathcal{J},u\in\mathcal{U}_j,w\geq 0}$. This is an auxiliary variable of the problem. The size of a state $|s_t|$ is the sum of its sub-states, and represents the total number of patients in the system at time t. The state space of all possible states is denoted by $\mathcal{S}$.
- The sub-action $a_{j_u,w,t} \in \mathbb{N}$ determines how many patients from sub-state $s_{j_u,w,t}$ to treat. The action a_t is a vector $(a_{j_u,w,t})_{j\in\mathcal{J},u\in\mathcal{U}_j,w\geq 0}$. This is the decision variable of the problem. The action space of all possible actions is denoted by $\mathcal{A}$. The set of possible actions given a state s_t is denoted by $\mathcal{A}(s_t)$.

The sets, parameters and variables described above are summarized in Table 6.1.

Table 6.1: Sets, variables, and parameters used for the model

Sets	
$\mathcal{T}$	Time periods
$\mathcal{J}$	Queues
$\mathcal{U}_j$	Urgency levels for queue $j \in \mathcal{J}$
$\mathcal{R}$	Resource types
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
Variables	
$a_{j_u,w,t}$	Sub-action: number of patients to treat at time $t \in \mathcal{T}$ from queue $j \in \mathcal{J}$ with priority $u \in \mathcal{U}_j$ who have been waiting for w time periods
$s_{j_u,w,t}$	Sub-state: number of patients in queue $j \in \mathcal{J}$ with priority $u \in \mathcal{U}_j$ who have been waiting for w time periods at time $t \in \mathcal{T}$
Parameters	
$\eta_{r,t}$	Available capacity of resource $r \in \mathcal{R}$ at time $t \in \mathcal{T}$
$\zeta_{j,r}$	Capacity of resource $r \in \mathcal{R}$ required by patient in queue $j \in \mathcal{J}$
q_{i_v,j_u}	Probability that a patient transfers from queue $i \in \mathcal{J}$ with priority $v \in \mathcal{U}_i$ to queue $j \in \mathcal{J}$ with priority $u \in \mathcal{U}_j$
$\lambda_{j_u,t}$	Number of new patients to enter queue $j \in \mathcal{J}$ with priority $u \in \mathcal{U}_j$ at time $t \in \mathcal{T}$
P	Planning horizon: the number of time periods to plan ahead

6.3. Objective

There are two objectives: minimize access time of patients relative to their urgency, and maximize the number of patients treated. The objective function should be a weighted combination of these two components.

6.3.1. Waiting time

The first objective can be expressed as minimizing a weighted sum of patients who are not treated on time. We define a cost function $c(j_u, w) : \mathcal{J} \times \mathcal{U} \times \mathbb{N} \to \mathbb{R}$ representing the cost of making a patient in queue j with urgency u wait for w time periods. If the waiting time has not yet exceeded the urgency level, the patient is not late yet and the cost should be zero: $c(j_u, w) = 0, \forall w < u$.

Furthermore, $c(j_u, w)$ should be non-decreasing in w (as higher waiting times should be penalized more heavily), and non-increasing in u (as a higher value of u indicates lower urgency). A possible cost function could be $c_{j_u,w} = \omega_j \cdot \frac{w}{u}\ \forall w \geq u$, to reflect the relative waiting time to urgency level. Here, ω_j is a parameter determining the weight of each queue relative to the other queues.

As a result, the first objective is given by:

$$\min \sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{U}_j} \sum_{w \geq 0} c(j_u, w) \cdot (s_{j_u,w,t} - a_{j_u,w,t}).$$

6.3.2. Patient treatment

The second objective is to maximize the number of patients treated. This should be done for three reasons, the first being the obvious health benefits patients experience from treatment. The second is that it is costly to leave resource capacity unused. The third is that efficiently using time and resource

capacity for treating more patients allows the surgeons to meet their strategic targets more easily.

A constraint requiring that all available resource capacity must be used to treat patients could lead to infeasibility. Therefore, a reward r_{j_u} is assigned for every patient treated from queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$. As a result, the second objective is given by:

$$\max \sum_{t\in\mathcal{T}} \sum_{j\in\mathcal{J}} \sum_{u\in\mathcal{U}_j} \sum_{w\geq 0} r_{j_u} \cdot a_{j_u,w,t}.$$

6.3.3. Contribution and value function

As a result of the above, the contribution function is:

$$C(s_t, a_t) = \sum_{j\in\mathcal{J}} \sum_{u\in\mathcal{U}_j} \sum_{w\geq 0} r_{j_u} \cdot a_{j_u,w,t} - c(j_u, w) \cdot (s_{j_u,w,t} - a_{j_u,w,t}). \tag{6.1}$$

Note that if the reward is set too low in comparison to the cost of waiting patients, then the optimal solution could be to accept very few new patients from the FC queue in order to enforce that patients never wait for too long. On the other hand, if it is set too high, it will become beneficial to have very long waiting lists so that there are always enough patients to fill available resource capacity. This trade-off must be considered carefully.

The objective is, given initial state $s \in \mathcal{S}$, to find an optimal policy π^* maximizing the value function:

$$V^*(s) = \mathbb{E}_{\pi^*}\left[\sum_{t\in\mathcal{T}} \gamma^t C(s_t, \pi(s_t)) | s_0 = s\right].$$

6.4. Planning ahead

Until this point, we have assumed in the base model that an action is selected at time t with perfect knowledge of the state at time t. However, as described in Section 3.2, planning decisions at SMK are made twelve weeks in advance, so that surgeons are informed of their schedules, and patients of their appointments, on time. This implies that at time t, we must determine action a_{t+6} (considering two-week time periods). To do so, some approximation of the state s_{t+6} must be made. This can be done using the current state s_t, and the actions $a_t, \dots, a_{t+5}$ that are taken in the meantime, as these actions have been chosen in previous time periods.

In the case of planning ahead, actions are only described by $a_{j,t} = \sum_{u\in\mathcal{U}_j} \sum_{w\geq 0} a_{j_u,w,t}$. This is also the way in which the rosters at SMK are expressed: only the appointment type (queue) is specified. One of the assumptions we made previously was that patients are booked into slots in the most efficient way. This means that we assume that, when selecting patients to book appointments for, the patient with the urgency level and waiting time for that appointment type resulting in the highest cost is chosen. In real life, there will be exceptions to this rule: the highest cost patient may not be available at the required time, appointments can be cancelled, and patients with higher urgency may arrive later on, when there are no available appointments. However, when comparing the quality of solution methods, it suffices to be consistent in this assumption.

We define the *planning horizon* as the number of time periods to plan ahead, and denote this by the parameter P. The procedure to determine the predicted state P time periods from time t, given actions $\{a_{j,t+p} : j \in \mathcal{J}, p = 0, \dots, P-1\}$, is provided in Algorithm 3.

Algorithm 3: Predicting a state P time periods ahead

Input: Current state s_t, number of time periods P, actions $\{a_{j,t+p} : j \in \mathcal{J}, p = 0, \dots, P-1\}$

for $p = 1, \dots, P$ **do**
  for $j \in \mathcal{J}$ **do**
    Sort the patients in queue j by cost
    for *the first* $a_{j,t}$ *patients with highest cost* **do**
      u = urgency level of patient;
      for $i \in \mathcal{J}$ **do**
        for $v \in \mathcal{U}_i$ **do**
          $s_{i_v,0,t+p}$ += q_{j_u,i_v} // Treat patient
        end
      end
    end
    for *the remaining patients* **do**
      u = urgency level of patient
      w = waiting time of patient
      $s_{j_u,w+1,t+p}$ += 1 // Patient must wait
    end
    for $u \in \mathcal{U}_j$ **do**
      $s_{j_u,0,t+p}$ += $\lambda_{j_u,t+p}$ // Add new patients
    end
  end
end
return s_{t+P}

6.5. Proposed solution methods

If we (conservatively) estimate that there are six urgency queues, the waiting times can be upper bounded by 13 (half a year), and the size of each resulting state $s_{j_u,w}$ can be upper bounded by 10, there are $10^{6 \cdot 13} = 10^{78}$ possible states. Clearly, this is far too large to allow for a lookup-table-based value function.

Since the state space is unmanageably large, the MDP should be solved using an approximation method. We choose to implement least-squares policy iteration, which was introduced in Section 4.1.2. To do so, patient pathways must be extracted from the available data and used to make a simulation. Next, basis functions must be chosen and tested, and effects of other algorithm parameters are investigated. For this purpose, test problems are made which require less time to solve than the full problem. Data from SMK is collected and analysed to determine the values of the parameters described in this chapter. Finally, the algorithm can be implemented using this data.

Next to approximate policy iteration, an (integer) linear program is formulated. The (I)LP requires an estimation of transition probabilities and a finite time horizon. However, it is likely to be much simpler and faster than LSPI, which makes it interesting to compare the approaches for practical use. Finally, simple greedy heuristics are developed. These approaches are compared in order to determine which delivers the best results. Here, we should not only consider the value of the contribution function, but also how the solution can be implemented in a hospital setting, for which simplicity and speed can outweigh small performance increases.

7

Solution methods

Now that we have translated the real-world problem to a mathematical model, we can identify various ways with which we can find (good) solutions to the problem. Since we have modelled the problem as a Markov decision process, the solution takes the form of a policy: a function or rule dictating which action to take given a state. An optimal policy maximizes the expected contribution gained by following that policy over time. Such an optimal policy can be found through dynamic programming, which we discuss in Section 7.1. However, as the real-world problem is very large, we cannot hope to find the optimal policy in reasonable time, and so we also investigate a number of approximation methods and heuristics. These are least-squares policy iteration (Section 7.2), linear programming (Section 7.3), decision rules (Section 7.4) and the static roster method used by SMK (Section 7.5).

7.1. Exact Value Iteration

Recall from Section 4.1.1 that the exact solution to an MDP can be found using dynamic programming. For realistic instances of the timeslot allocation problem at SMK, this will not be possible, as the state and action spaces are very large, and the outcome space is large and difficult to calculate- the three curses of dimensionality all apply. However, it is useful to find the exact solution to very small test instances, in order to compare the performance of the other solution methods discussed here. We also use this to compare different basis functions that can be used for least-squares policy iteration, which is explained later on.

Finding an exact solution requires solving the Bellman equation:

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right), \forall s \in \mathcal{S}.$$

This can be done using the exact value iteration (EVI) algorithm for infinite horizon optimization provided in [12], shown in Algorithm 4.

Equation (7.1) in the algorithm presents us with one more problem: calculating $P(s'|s,a)$. An action causes patients to leave and enter queues with given probabilities, but calculating the probability that all these transitions add up to a given state turns out to be quite difficult.

A much easier approach, giving an equivalent solution, is not to sum over all possible next states, but over all possible realizations of transitions caused by action a. Let $I = \{(i,j) : q_{ij} > 0, i,j \in \mathcal{J} \cup \{0\}\}$ be the pairs of indices of queues between which patients can transition, including the 'outside' queue 0. Now let $X = (X_{i,j})_{(i,j)\in I}$ be a random variable denoting the number of patients transitioning from queue i to queue j. Let a_i be the action indicating the number of patients to treat from queue $i \in \mathcal{J}$. Then X_{ij} is binomially distributed: $X_{ij} \sim B(a_i, q_{ij})$. Let $\mathcal{X}(a)$ be the set of possible realizations of transitions

Algorithm 4: EVI algorithm for infinite horizon problems [12]

Initialise $v^0(s) = 0 \; \forall s \in \mathcal{S}$
Choose a tolerance parameter $\epsilon > 0$
Choose a discount factor $\gamma \in (0, 1)$
Let $\delta = \epsilon(1-\gamma)/2\gamma$
Let $\Delta = \delta + 1$
Set $n = 1$
while $\Delta \geq \delta$ **do**
  for $s \in \mathcal{S}$ **do**

$$v^n(s) = \max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v^{n-1}(s') \right) \tag{7.1}$$

  end
  $\Delta = \left\| v^n - v^{n-1} \right\|$
  $n = n + 1$
end
return $V^* = v^n$

resulting from action a:

$$\mathcal{X}(a) = \left\{ x \in \mathbb{Z}^{|I|} : x_{ij} \geq 0 \; \forall (i,j) \in I, \sum_{j \in \mathcal{J}} x_{ij} = a_i \; \forall i \in \mathcal{J} \right\}.$$

Recall the transition function $T(s,a,x)$ introduced in Chapter 4.1 providing the new state given previous state s, action a and (a realization of) exogenous information x. Equation (7.1) can now be written as:

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \sum_{x \in \mathcal{X}(a)} P(X = x|a) V^*(T(s,a,x)) \right) \\ &= \max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \sum_{x \in \mathcal{X}(a)} \left(\prod_{(i,j) \in I} P(X_{ij} = x_{ij}|a_i) \right) V^*\left(T(s,a,x)\right) \right) \\ &= \max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \sum_{x \in \mathcal{X}(a)} \left(\prod_{(i,j) \in I} \binom{a_i}{x_{ij}} q_{ij}^{x_{ij}} (1 - q_{ij})^{a_i - x_{ij}} \right) V^*\left(T(s,a,x)\right) \right). \end{aligned}$$

In [12], Powell suggests using a Gauss-Seidel variation in order to speed up convergence. This exploits the fact that during an iteration, $v^n(s')$ is already known for states previously visited within the iteration (denoted by $s' < s$), and can be used instead of v^{n-1} in the updating equation for $v^n(s)$.

The algorithm using the Gauss-Seidel variation and adaptation of Equation (7.1) is shown in Algorithm 5. Within each iteration, we must loop over all states. For each state, a maximum is taken over the entire action space. Finally, we sum over all possible transitions for each action, and then multiply over all possible transitions. The resulting running time of the algorithm is $\mathcal{O}(n \cdot |\mathcal{S}| \cdot |\mathcal{A}| \cdot |\mathcal{X}|)$. The size of the transition space, $|\mathcal{X}|$, is exponential in the size of the action space, $|\mathcal{A}|$.

Algorithm 5: Adapted EVI algorithm with Gauss-Seidel variation

Initialise $v^0(s) = 0\ \forall s \in \mathcal{S}$
Choose a tolerance parameter $\epsilon > 0$
Let $\delta = \epsilon(1-\gamma)/2\gamma$
Set $n = 1$
while $\left\|v^n - v^{n-1}\right\| \geq \delta$ **do**
  for $s \in \mathcal{S}$ **do**

$$v^n(s) = \max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \sum_{\substack{x \in \mathcal{X}(a): \\ T(s,a,x) < s}} \left(\prod_{(i,j) \in I} \binom{a_i}{x_{ij}} q_{ij}^{x_{ij}} (1-q_{ij})^{a_i - x_{ij}} \right) v^n(T(s,a,x)) \right.$$
$$\left. + \gamma \sum_{\substack{x \in \mathcal{X}(a): \\ T(s,a,x) \geq s}} \left(\prod_{(i,j) \in I} \binom{a_i}{x_{ij}} q_{ij}^{x_{ij}} (1-q_{ij})^{a_i - x_{ij}} \right) v^{n-1}(T(s,a,x)) \right)$$

  end
  $n = n + 1$
end
return $V^* = v^n$

7.2. Least-Squares Policy Iteration

As exact value iteration will not work for realistic problem sizes, we consider an approximation method which can be implemented on the full problem. This is least-squares policy iteration, which was introduced in Section 4.1.2. The algorithm learns a policy by learning a parametrised approximation of the value function, $\overline{V}(s) = \theta^\top \phi(s)$. Since this is an approximation method, LSPI is hindered much less by the curses of dimensionality, and can therefore be used to solve larger instances of the timeslot allocation problem.

We propose a combination of the off-policy LSPI-Q (Algorithm 1) and on-policy LSPI-V (Algorithm 2) algorithms presented in Section 4.1.2, giving the off-policy LSPI-V algorithm shown in Algorithm 6. To the best of our knowledge, this version has not been used yet in the literature. For this problem, the value function version must be used as the action space is relatively large and defined by a difficult resource constraint, so iterating over enough (s, a) samples to obtain a good approximation of the Q-function requires a much larger dataset than using only state samples to approximate the value function. The off-policy variant is used, because implementation of on- and off-policy variants quickly showed that the on-policy variant did not converge. Finally, we use the least-squares temporal differencing update (as opposed to recursive LSTD) as F is small enough for this problem to efficiently compute the inverse of A_M. This avoids many matrix-vector multiplications in the inner loop, and reduces numerical errors.

7.2.1. Simulation of the environment

LSPI requires a simulation from which a dataset of states can be sampled. The simulation is also used to test and compare the other solution methods. The simulation should mirror the way in which patients transition between queues following consultations and surgery, as described in Section 2.1. This can be done by extracting patient pathways from data provided by SMK. The way in which this is done is described in Section 8.3.

A patient p can be formulated as a tuple $\langle v, i, w \rangle$ where v is a vector of appointment types which must be visited in that order, i is a counter indicating at which stage they are of their treatment (starting at 1),

Algorithm 6: Off-policy LSPI-V

$\theta_0 = \mathbf{0} \in \mathbb{R}^F$
Choose small constants $\epsilon, \delta > 0$
Generate a dataset of states $D = \{s_1, \dots, s_M\}$
Define the policy:

$$\pi(s|\theta) = \arg\max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \theta^\top \phi(\overline{T}(s,a)) \right)$$

while $||\theta_n - \theta_{n-1}|| \geq \delta$ **do**
  $b_0 = \mathbf{0} \in \mathbb{R}^F$
  $A_0 = \epsilon I$, using $F \times F$ identity matrix I
  for $m = 1, \dots, M$ **do**
    Compute action $a_m = \pi(s_m|\theta_{n-1})$
    Obtain a sample of exogenous information x_m from the simulation
    Determine the next state $s'_m = T(s_m, a_m, x_m)$
    Perform the LSTD update:

$$b_m = b_{m-1} + C(s_m, a_m)\phi(s_m)$$
$$A_m = A_{m-1} + \phi(s_m) \cdot (\phi(s_m) - \gamma\phi(s'_m))^\top$$

  end
  $\theta_n = A_M^{-1} b_M$
end
return *the approximate policy* $\pi(s|\theta_N)$

and w is their current waiting time. For example $p = \langle (FC_4, RC_6, RC_3, OR_4, DC_3), 4, 2 \rangle$ indicates a patient who has been waiting for surgery with urgency level 4 for two time periods after completing one first consultation and two repeat consultations. When the patient is 'treated', i is incremented by 1 and w is set to 0. If a patient must wait, w is incremented by 1. When $i > |v|$ the patient has completed their treatment and is removed from the system. We consider the actual patient paths as unknown to the algorithm; these are only used to determine the next state after performing an action.

7.2.2. Action selection

Using the policy function to choose an action requires solving:

$$\pi(s|\theta) = \arg\max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \theta^\top \phi(\overline{T}(s,a)) \right),$$

where $\overline{s} = \overline{T}(s,a)$ is an approximation of the next state. The space of feasible actions $\mathcal{A}(s)$ is constrained by time capacity and queue length. If the basis functions are non-linear, or a non-linear parametrisation is used for the approximate value function, the action can be selected by calculating the value for all possible actions and choosing the maximum. However, when we use linear basis functions and a linear parametrisation, it is faster to solve the equation using a linear program.

We now arrive at the problem of formulating an approximate transition function. Hulshof uses the post-decision state in his implementation of approximate dynamic programming in [9], following the recommendation of Powell in [12]. The post-decision state is the state where patients who are treated have been removed from their queues, and the waiting time of untreated patients is incremented by one time unit. We will argue why not the post-decision state, but the expectation of the following state, can and should be used.

When an MDP can be solved exactly, given optimal value function V^*, the optimal policy can be

described as:

$$\pi(s) = \arg\max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right)$$
$$= \arg\max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \mathbb{E}[V^*(s')|s,a] \right).$$

Now, rather than an optimal value function, we have an approximate value function $\overline{V}$. Since we are using a linear parametrisation and linear basis functions, this function is linear in the state vector. Therefore, we can rewrite $\mathbb{E}[\overline{V}(s')|s,a]$ as $\overline{V}(\mathbb{E}[s'|s,a])$. This gives:

$$\overline{\pi}(s) = \arg\max_{a \in \mathcal{A}(s)} \left(C(s,a) + \gamma \theta^\top \phi(\mathbb{E}[s'|s,a]) \right).$$

For this problem, the expected next state can be calculated quite easily. The only stochastic information are the patient transitions and, possibly, arrivals. Given an action a, the expected number of patients transitioning from queue i with urgency v to queue j with urgency u is simply $\sum_{w \geq 0} q_{i_v, j_u} \cdot a_{i_v, w}$. This does require us to estimate the transition probabilities q from data. If patient arrivals are stochastic, the expected number of patients to arrive to each queue, λ_{j_u}, should also be estimated from data. The rest of the state transition is determined by deterministic events: patients leave the queue if they are treated, or wait on extra time period if they are not.

Using the above, we can formulate the following linear program to determine the next action using the expected next state, which we refer to as the *policy-LP*:

$$\max_a \quad C(s,a) + \gamma \sum_{f=1}^{F} \theta_f \cdot \phi_f(\overline{s}) \tag{7.2a}$$

s.t.

$$\overline{s}_{j_u,0} = \lambda_{j_u} + \sum_{i \in \mathcal{J}} \sum_{v \in \mathcal{U}_i} \sum_{w=0}^{W} q_{i_v, j_u} \cdot a_{i_v, w}, \quad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, \tag{7.2b}$$

$$\overline{s}_{j_u,w} = s_{j_u,w-1} - a_{j_u,w-1}, \quad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, 1 \leq w \leq W-1, \tag{7.2c}$$

$$\overline{s}_{j_u,W} = \sum_{w=W-1}^{W} s_{j_u,w} - a_{j_u,w}, \quad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, \tag{7.2d}$$

$$a_{j_u,w} \leq s_{j_u,w}, \quad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, 0 \leq w \leq W, \tag{7.2e}$$

$$\sum_{j \in \mathcal{J}} \zeta_{j,r} \sum_{u \in \mathcal{U}_j} \sum_{w \geq 0} a_{j_u,w} \leq \eta_r, \quad \forall r \in \mathcal{R}, \tag{7.2f}$$

$$a_{j_u,w} \in \mathbb{Z}^+, \quad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, 0 \leq w \leq W. \tag{7.2g}$$

The contribution function $C(s,a)$ is as defined in Equation (6.1). The first three constraints define the expected next state, which is used in the objective function, while the last three constraints define all feasible actions. Constraint (7.2b) defines the expectation of how many patients will transition to new queues given an action a, and the expected number of new patients λ. This is the only place where the expected next state differs from the actual next state, where we observe a simulated realization of patient transitions for an action. Constraints (7.2c) and (7.2d) ensure that patients who are not treated stay in the same queue and their waiting time is incremented by 1 time unit, unless their waiting time has reached the upper bound, in which case it stays the same. Constraint (7.2e) ensures that we cannot treat more patients than the number of patients in a queue, and the resource constraint (7.2f) ensures that the action does not exceed the resource capacity.

There is an integer constraint on the actions, (7.2g), as it is only possible to treat a whole number of patients. However, since this program must be solved for every state in the dataset, in every iteration of the algorithm, it is wise to relax the integer constraint to a non-negativity constraint in order to reduce

the solving time. In that case, the actions resulting in the optimal solution should be rounded down to the nearest integer to ensure feasibility; rounding up may violate constraints (7.2e) or (7.2f). Rounding down could result in some loss of optimality, as some available capacity could go unused. When the policy is actually used for planning, the integer constraint can and should be used, as in that case the policy-LP only needs to be solved once per planning period.

7.2.3. Basis functions

Achieving a good state representation through well-chosen basis functions is essential to the convergence and performance of LSPI. In this section, various basis functions are proposed. We also introduce a linear regression method to find a corresponding parameter vector θ^* for each basis function using an exact solution to the problem.

7.2.3.1. Proposed functions

Basis functions may be non-linear (for example, Gaussian or quadratic functions of features), but we focus on linear functions for simplicity. This also ensures that action selection can be done by solving the policy-LP, and that $\mathbb{E}[\phi(s)] = \phi(\mathbb{E}[s])$.

When designing basis functions, it is important to take into account the dimension F. Increasing F will increase the computation time, as the algorithm requires computing the inverse of $F \times F$ matrix A. A high-dimensional parametrisation also increases the danger of overfitting. On the other hand, decreasing F may result in a poor approximation of the value function as the features become too general to accurately represent the state space.

In Table 7.1, some possible basis functions are proposed. The notation $U = \sum_{j \in \mathcal{J}} \mathcal{U}_j$ is used for simplicity. Basis functions 1 and 4 are similar to those used by Hulshof in [10].

Table 7.1: Possible basis functions for least-squares policy iteration

	Feature	**Basis function**	F
1.	Number of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$ who have been waiting for w time periods	$s_{j_u,w}, \forall j \in \mathcal{J}, u \in \mathcal{U}_j, 0 \leq w \leq W$	$(W+1) \cdot U$
2.	Number of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$ who are early for treatment, who are on time for treatment, and who are late	$\sum_{w<u-1} s_{j_u,w}, \sum_{u-1 \leq w \leq u} s_{j_u,w}, \sum_{w>u} s_{j_u,w}, \forall j \in \mathcal{J}, u \in \mathcal{U}_j$	$3 \cdot U$
3.	Cost of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$	$\sum_{w>u} c(j_u, w) \cdot s_{j_u,w}, \forall j \in \mathcal{J}, u \in \mathcal{U}_j$	U
4.	Total number of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$	$\sum_{w \geq 0} s_{j_u,w}, \forall j \in \mathcal{J}, u \in \mathcal{U}_j$	U

The dimension of the first basis function, although much smaller than the state space, may be too large as it is a multiple of the upper bound W on patient waiting times. This also forces us to upper bound the waiting times, while this is not strictly necessary for the other basis functions. However, an upper bound on waiting times is necessary in any case for the policy-LP, which becomes slower to solve if the upper bound is increased. Since $U = 9$ for the full problem, the other basis functions should be

manageable. It is also possible to use a combination of multiple basis functions, for example, the cost of each urgency queue *and* the total number of patients in each urgency queue.

An intercept term of 1 could be added to each basis function, increasing the dimension by one. If we do not do this, for example, the all-zero state will always have value $V(\mathbf{0}) = \theta^\top \phi(\mathbf{0}) = 0$, which will not necessarily correspond with the exact value for that state. The intercept will improve the ability to approximate the exact value function. However, it does not change the resulting policy, as it only adds a constant term to the approximate value function and therefore does not affect which action results in the maximum value.

7.2.3.2. Linear regression

If the exact value function V^* is known, we can use it to determine which basis functions should be used for LSPI. Given a basis function ϕ, linear regression can be used to calculate the θ minimizing the mean squared error (MSE) between the exact and approximate solutions for a training set of n states $\{s_1, \dots, s_n\} \subset \mathcal{S}$ [12]:

$$\theta^* = \min_{\theta \in \mathbb{R}^F} \left(\sum_{i=1}^{n} \left(V^*(s_i) - \theta^\top \phi(s_i) \right)^2 \right). \tag{7.3}$$

Define the input matrix $X \in \mathbb{R}^{n \times F}$, providing the input data for each state, and the target vector $Y \in \mathbb{R}^n$, providing the target value for each state, as follows:

$$X = \begin{pmatrix} \phi_1(s_1) & \dots & \phi_F(s_1) \\ \vdots & \ddots & \vdots \\ \phi_1(s_n) & \dots & \phi_F(s_n) \end{pmatrix}, \qquad Y = \begin{pmatrix} V^*(s_1) \\ \vdots \\ V^*(s_n) \end{pmatrix}.$$

Then, θ^* solving Equation (7.3) can be found by calculating:

$$\theta^* = (X^\top X)^{-1} X^\top Y.$$

More background on linear regression can be found in [34].

After using a training set of n states to find θ for a given basis function, the performance of that basis function can be measured by calculating the MSE on a test set of m states:

$$\text{MSE}(\phi, \{s_1, \dots, s_m\}) = \sum_{i=1}^{m} \left(V^*(s_i) - \theta^{*\top} \phi(s_i) \right)^2.$$

The training and test set should be disjoint, as the test set measures how well the parametrisation found can generalise to unseen samples. Furthermore, k-fold cross-validation can be used to obtain an accurate estimation of the true MSE for the whole dataset [34]. In this technique, we split the dataset into k distinct subsets. Then, $k - 1$ subsets are combined and used as the training set, while the remaining subset is used as the test set. This is repeated k times until each subset has been used as a test set once. The average MSE of these k tests provides our estimation of the true MSE. Finally, this process should be repeated for different basis functions. The one resulting in the lowest MSE provides the closest approximation to the exact value function.

7.3. (Integer) Linear Programming

This section presents the (integer) linear program which can be used to solve the deterministic, finite-time-horizon variant of the MDP presented in Chapter 6. The same sets, parameters, and variables are used; a summary can be found in Table 6.1.

Some changes must be made to the base model as an (I)LP is deterministic and cannot handle an infinite time horizon. Therefore, the program uses transition fractions q_{i_v, j_u} rather than transition probabilities. For example, $q_{RC_3, OR_4} = 0.1$ implies that 10% of patients in the RC queue with urgency level 3

will transfer to the OR queue with urgency level 4 when they are treated. In this way, all stochasticity is removed from the problem. Finally, rather than $\mathcal{T} = \mathbb{N}$ we use a finite time horizon $\mathcal{T} = \{0, 1, \dots, T\}$ for some $T \in \mathbb{N}$.

7.3.1. Constraints and objective function

The total number of patients to treat from a queue is given by:

$$a_{j_u,t} = \sum_{w \geq 0} a_{j_u,w,t}, \qquad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, t \in \mathcal{T}. \tag{7.4}$$

The newly arrived number of patients in a queue at any time is the sum of the external arrivals to that queue and patients transitioning from other queues to that queue as a result of treatment:

$$s_{j_u,0,t} = \lambda_{j_u,t} + \sum_{i \in J} \sum_{v \in \mathcal{U}_i} q_{i_v,j_u} a_{i_v,t}, \qquad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, t \in \mathcal{T}. \tag{7.5}$$

Patients who are not treated remain in the same queue, and their waiting time is incremented by one time period:

$$s_{j_u,w,t} = s_{j_u,w-1,t-1} - a_{j_u,w-1,t-1}, \qquad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, w > 0, t \in \mathcal{T}. \tag{7.6}$$

It is not possible to treat more patients than the number of patients currently in that queue:

$$a_{j_u,w,t} \leq s_{j_u,w,t}, \qquad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, w \geq 0, t \in \mathcal{T}. \tag{7.7}$$

The resources required are constrained by the available resource capacity:

$$\sum_{j \in \mathcal{J}} \zeta_{j,r} \sum_{u \in \mathcal{U}_j} a_{j_u,t} \leq \eta_{r,t}, \qquad \forall r \in \mathcal{R}, t \in \mathcal{T}. \tag{7.8}$$

It is only possible to treat an integer number of patients:

$$a_{j_u,w,t} \in \mathbb{N}, \qquad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, w \geq 0, t \in \mathcal{T}. \tag{7.9}$$

To make the problem faster to solve, this integer constraint could be relaxed to $a_{j_u,w,t} \geq 0$. In that case, the actions in the optimal solution must be rounded down to the nearest integer to ensure feasibility, as in the policy-LP. Again, this could result in some loss of optimality, and should therefore not be done in the event that the ILP is actually used for planning.

Finally, the initial state s_0 should be fixed to the state for which the first planning decision is made.

Since the contribution function $C(s_t, a_t)$ is linear in the state and action variables, it can be used for the objective function of the (I)LP. Therefore, the objective function is:

$$\max_{a \in \mathcal{A}^T} \sum_{t \in \mathcal{T}} \gamma^t C(s_t, a_t)\,.$$

7.3.2. Possible additional constraints

The constraints above define a feasible solution for the planning problem. Additional constraints can be defined which reflect wishes or rules of the capacity planning department. These constraints could also be used in LSPI, by incorporating them in the policy-LP. Some possibilities are:

The OR queue must always be longer than some fixed minimum length l:

$$\sum_{u \in \mathcal{U}_{OR}} \sum_{w \geq 0} s_{OR_u,w,t} \geq l, \qquad \forall t \in \mathcal{T}. \tag{7.10}$$

The number of new patients admitted over m time periods should exceed a minimum value p:

$$\sum_{t=t'}^{t'+m} \sum_{u \in \mathcal{U}_{FC}} a_{FC_u,t} \geq p, \qquad \forall t' \in \mathcal{T}. \tag{7.11}$$

Patients cannot be treated more than two weeks before the deadline indicated by their urgency level:

$$a_{j_u,w,t} = 0, \qquad \forall j \in \mathcal{J}, u \in \mathcal{U}_j, 0 \leq w < u - 1, t \in \mathcal{T}. \tag{7.12}$$

Since it is more costly to make patients with higher waiting times wait for treatment, we may assume that in the optimal solution, these patients are always treated first. Therefore, the following are valid inequalities (where K is some high number):

$$\begin{aligned} s_{j_u,w,t} - a_{j_u,w,t} &\leq 1 - \epsilon + L y_{j_u,w,t}, && \forall j \in \mathcal{J}, u \in \mathcal{U}_j, w > 0, t \in \mathcal{T}, \\ a_{j_u,w-1,t} &\leq K(1 - y_{j_u,w,t}), && \forall j \in \mathcal{J}, u \in \mathcal{U}_j, w > 0, t \in \mathcal{T}, \\ y_{j_u,w,t} &\in \{0,1\}, && \forall j \in \mathcal{J}, u \in \mathcal{U}_j, w > 0, t \in \mathcal{T}. \end{aligned}$$

This ensures that all patients with higher waiting times are treated before patients with lower waiting times within the same queue and urgency level: $a_{j_u,w-1,t}$ can only be positive when all patients with a higher waiting time have been treated, i.e., when $s_{j_u,w,t} = a_{j_u,w,t}$. Adding this inequality to the linear program reduces the size of the feasible region, which could reduce the time taken to solve the program. However, if the integer constraint on $a_{j_u,t}$ is relaxed, adding the binary constraint on $y_{j_u,w,t}$ again makes it a mixed integer linear program. This can significantly increase solving time.

7.3.3. Rolling time horizon

The solution to the LP is a vector $a = (a_0, \dots, a_T)$ of actions to perform over a time horizon of length T. However, actually performing these actions may not lead to good results in the real world for two reasons. The first is that, since the solution is fixed and transition fractions are used rather than probabilities, the solution is not robust and cannot respond when the real-world situation strays from its expected course. The second reason is that the finite time horizon distorts the solution towards the end, as future consequences of those actions are not taken into account.

It is more realistic that a capacity planning department making use of this LP would solve it every time a planning decision should be made, with s_0 as the (predicted) state for which a schedule must be made, and only implement the first action a_0. T should still be set to a longer period of time, such as one year, so that future consequences of that first action are taken into account. We refer to this as a rolling time horizon. In the case of SMK, this would imply solving the LP once every two weeks with $s_0 = s_{t+6}$, the predicted state twelve weeks from now, and scheduling the number of FC, RC and DC appointments indicated by a_0.

7.4. Decision rules

In this section, we formulate a number of decision rules that can provide a feasible solution to the timeslot allocation problem. The following decision rules are considered:

- **Highest Contribution:** Treat the patient from queue $j \in \mathcal{J}$ with urgency $u \in \mathcal{U}_j$ and waiting time w that would result in the highest increase in contribution if they are treated now. This is expressed as the sum of the reward and the cost of the patient; by treating that patient, we gain the associated reward and avoid the associated cost:

  $$(j,u,w) = \arg\max \{c(j_u,w) + r_{j_u} : j \in \mathcal{J}, u \in \mathcal{U}_j, w \geq 0, s_{j_u,w} \geq 1\}.$$

  Continue until all resource capacity has been used or the queues are empty.

- **Highest Cost Queue:** Treat a patient from the queue $j \in \mathcal{J}$ and urgency $u \in \mathcal{U}_j$ with the highest total cost, i.e.,

  $$(j,u) = \arg\max \left\{ \sum_{w \geq 0} c(j_u,w) \cdot s_{j_u,w} : j \in \mathcal{J}, u \in \mathcal{U}_j \right\}.$$

The patient in the selected queue with the highest waiting time is treated. Recalculate the total queue costs after selecting the patient, and continue until all resource capacity has been used or the queues are empty.

- **Longest Queue:** Treat the patient from the longest queue $j \in \mathcal{J}$ with urgency $u \in \mathcal{U}_j$, i.e.,

$$(j,u) = \arg\max\left\{\sum_{w\geq 0} s_{j_u,w} : j \in \mathcal{J}, u \in \mathcal{U}_j\right\}.$$

The patient in the selected queue with the highest waiting time is treated. Continue until all resource capacity has been used or the queues are empty.

- **Split Cost:** Calculate the total cost of each queue $j \in \mathcal{J}$ and urgency level $u \in \mathcal{U}_j$. For each resource, distribute the available resource capacity over each queue, proportional to the relative cost of that queue to the total cost of all queues requiring the same resource:

$$a_{j_u} = \min\left(s_{j_u}, \left\lfloor \eta_r \cdot \frac{\sum_{w\geq 0} c(j_u,w)\cdot s_{j_u,w}}{\sum_{j\in\mathcal{J}_r}\sum_{u\in\mathcal{U}_j}\sum_{w\geq 0} c(j_u,w)\cdot s_{j_u,w}} \right\rfloor\right), \qquad \forall j \in \mathcal{J}, u \in \mathcal{U}_j,$$

where r is the resource required by queue j, and $\mathcal{J}_r$ is used to denote the set of queues also requiring resource r. Within each queue, treat patients with the highest waiting time first. Note that this decision rule only works when queues require only one resource, as is the case for this problem.

These rules are simple and easy to implement and understand. For hospitals, this is beneficial as complex optimization software is not required and a solution can be found quickly. Also, it is much easier to explain to hospital staff why certain decisions should be made. The rule itself provides an explanation- for example, “we should always treat our most urgent patients first”. This is in contrast with the other approaches described, which are essentially black-boxes, as we cannot easily argue why the solution provided is optimal. However, we expect that this increase in simplicity and explainability comes with a loss of performance. In Section 9.2.3, we compare the decision rules with each other on a large test problem, and in Section 9.2.4, the best-performing rule is compared with the other solution methods.

7.5. Static rosters

Finally, we should consider the planning method currently used by SMK. The hospital generally does not make adaptations to the number of FC, RC, and DC appointments that are planned within an OD session. Rather, a static roster is used with a fixed number of each appointment type, as this can be done quickly and easily. Manually adapting every surgeon’s roster every other week is much more work, and is only done when the capacity planning department determines that an intervention must be done to prevent extremely long or short waiting lists. However, if we find that a significant improvement can be obtained using one of the solution methods explored in this thesis compared to the static rosters, it is worth looking into the implementation of dynamic rosters.

For most surgeons, there are three different static rosters that can be used to plan an OD session, for when the surgeon is aided by zero, one or two assistants. Schedules for a session with more assistants allow for more OD appointment slots. However, the proportion of FC, RC and DC appointments within one session remains (roughly) the same. The hospital determines these proportions by calculating the ratios between the appointment types for previous patients of the surgeon, and total OD time. Under the assumption that future patients and care pathways are similar to those in the past, this should ensure that waiting lists remain relatively balanced. We apply this method as well, in order to provide an accurate comparison between the new solution methods discussed in this thesis and the current planning method of SMK.

8

Data analysis

In order to test how well the solution methods from Chapter 7 perform on the model constructed in Chapter 6, we must artificially generate data, and collect real-world data, with which the model parameters can be defined. The data used is summarized in this chapter. We generate two datasets artificially: one for a small test problem (Section 8.1), and one for a large test problem (Section 8.2). Then, we analyse real-world data provided by SMK in order to define the parameters of the full timeslot allocation problem (Section 8.3).

The two test problems are required for a number of reasons. First, the small test problem is required as we need to have a problem which is solvable using exact value iteration. This is used to compare how well a policy obtained through LSPI can approximate the optimal policy found through EVI. Therefore, the small test problem should have a state space small enough to permit an exact solution. Secondly, LSPI requires some parameter tuning, and for this purpose it is desirable to have a test problem which is similar to the real-world problem, but not quite as large so that it is faster to solve and experiments can be performed efficiently. Due to the strict size constraint, the small test problem is too different from the real-world problem, so a second, large test problem is defined.

8.1. Small test problem

The small test problem is solved using exact dynamic programming, which requires looping over all possible states. In any case, the size of each sub-state must be upper bounded by some value l_j, i.e., $s_{j_u,w,t} \leq l_j$, and the possible waiting times must be upper bounded by some value W, i.e., $0 \leq w \leq W$. Without these upper bounds, the size of the state space would be infinite. In order to solve the test problem within reasonable time, a test problem is generated for which the state space is of a manageable size.

For each queue $j \in \mathcal{J}$, there are $|\mathcal{U}_j| \cdot (W+1)$ corresponding sub-states (the "+1" arises since 0 is a possible waiting time). Each sub-state can take on $l_j + 1$ possible values (again, the "+1" arises since 0 is a possible sub-state size). Therefore, size of the state space is given by

$$|\mathcal{S}| = \Pi_{j \in \mathcal{J}} (l_j + 1)^{|\mathcal{U}_j| \cdot (W+1)}.$$

It is clear that $|\mathcal{S}|$ grows exponentially in the number of queues, number of urgency levels, and the upper bound on the waiting time.

The following parameters are used for the small test problem:

- There are two queues: $\mathcal{J} = \{RC, OR\}$. The RC queue requires one unit of OD time per patient and the OR queue requires one unit of OR time per patient.
- Each time period represents a planning period of two weeks.
- The RC queue has only one urgency level, and the OR queue has two: $\mathcal{U}_{RC} = \{1\}$ and $\mathcal{U}_{OR} = \{0, 1\}$.

- The possible waiting times are 0, 1 and 2, so the upper bound on waiting times, W, is 2. If a patient with waiting time 2 must wait an extra time period, their waiting time is kept at 2.
- The queue lengths are upper bounded by $l_{RC} = 7$ and $l_{OR} = 2$.
- As a result, a state s is given by the vector:

$$s = (s_{RC_1,0}, s_{RC_1,1}, s_{RC_1,2}, s_{OR_0,0}, s_{OR_0,1}, s_{OR_0,2}, s_{OR_1,0}, s_{OR_1,1}, s_{OR_1,2})$$

  This gives $8^{1\cdot3} \cdot 3^{2\cdot3} = 373,248$ possible states. For illustration, if we were to increment W by 1, this would result in a state space of size $26,873,856$.
- Resource capacities are set to $\eta_{OD} = 4$ and $\eta_{OR} = 2$ for all time periods.
- Transition probabilities are $q_{RC_1,RC_1} = 0.3, q_{RC_1,OR_0} = 0.2, q_{RC_1,OR_1} = 0.3, q_{RC_1,Exit} = 0.2, q_{OR_0,RC_1} = 0.4, q_{OR_0,Exit} = 0.6, q_{OR_1,RC_1} = 0.4$ and $q_{OR_1,Exit} = 0.6$. All other transition probabilities are 0. The possible transitions are shown in Figure 8.1.
- Three new patients enter the RC queue in every time period, so $\lambda_{RC} = 3$ and all others are 0.
- Actions a_{j_u} indicate how many patients to treat out of each urgency queue. As before, actions are bounded by the queue size and resource capacity. We assume that patients are treated in order of highest waiting time, as this minimizes cost and does not result in different expected transitions. The value $a_{j_u,w}$ required for the contribution function can be calculated accordingly. Ignoring waiting times in the action results in a much smaller action space, and therefore, shorter computation time.
- The cost for keeping a patient waiting is $c_{j_u,w} = \omega_j \cdot \frac{w}{u+1}$ where $\omega_{RC} = 1$ and $\omega_{OR} = 4$. Adding 1 to u in the denominator is necessary here as one of the urgency levels is 0. The rewards for treating patients are $r_{RC} = 1$ and $r_{OR} = 4$. The contribution function from Equation (6.1) is used.

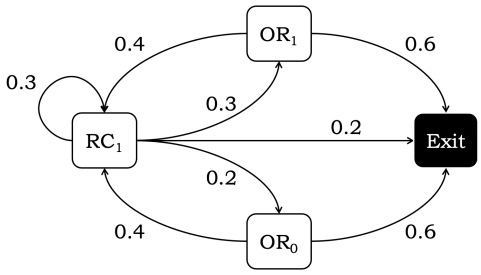

Figure 8.1: Queues and transition probabilities for the small test problem

8.2. Large test problem

In this section, we define a large test problem on which we can test different parameter settings and algorithm variations for the solution methods. The test problem should be a scaled-down version of the full problem, so that the full problem is well-represented but experiments can still be done efficiently. Since exact value iteration is not used to solve this problem, we no longer need to set an upper bound on the sub-state sizes. However, an upper bound on the waiting times is required to define the range of waiting times in the policy-LP in LSPI, and for the rolling-horizon LP. The higher this upper bound is set, the longer the linear programs will take to solve.

The following parameters are used for the large test problem:

- There are four queues: $\mathcal{J} = \{FC, RC, OR, DC\}$. The FC, RC and DC queues require one unit of OD time and the OR queue requires one unit of OR time per patient.
- The FC, RC and DC queues have one urgency level, and the OR queue has two: $\mathcal{U}_{FC} = \{2\}, \mathcal{U}_{RC} = \{4\}, \mathcal{U}_{OR} = \{2, 4\}$ and $\mathcal{U}_{DC} = \{3\}$.

- The upper bound on waiting times is different per queue, dependent on the urgency level: $W_{j_u} = 3 \cdot u, \forall j \in \mathcal{J}, u \in \mathcal{U}_j$. This allows higher waiting times for queues in which patients generally wait longer due to higher urgency levels, without increasing the running time of the linear programs too much.

- When generating patients for a random state, their waiting times are drawn from an exponential distribution with scale parameter $\beta = u$, where u is the patient's urgency level, and rounded to the nearest integer below W_{j_u}.

- Resource capacities are set to $\eta_{OD} = 16$ and $\eta_{OR} = 2$.

- Transition probabilities are shown in Table 8.1. In each row, the probability of a patient transferring from the appointment type in that row to the appointment type indicated in each column is shown. The 'Exit' column is the probability that the patient will leave the system.

Table 8.1: Transition probabilities for the large test problem

	FC_2	RC_4	OR_2	OR_4	DC_3	Exit
FC_2	0	0.5	0.01	0.1	0	0.39
RC_4	0	0.4	0.02	0.15	0	0.33
OR_2	0	0.2	0	0	0.75	0.05
OR_4	0	0.25	0	0	0.7	0.05
DC_3	0	0.6	0	0	0	0.4

- Eight new patients enter the FC queue in every time period, so $\lambda_{FC} = 8$ and all others are 0.

- The cost for keeping a patient waiting is $c_{j_u,w} = \omega_j \cdot \frac{w}{u+1}$ where $\omega_{FC} = 2, \omega_{RC} = 1, \omega_{OR} = 4$ and $\omega_{DC} = 1$. The rewards for treating patients are $r_{FC} = 2, r_{RC} = 2, r_{OR} = 10$ and $r_{DC} = 1$. The contribution function from Equation (6.1) is used.

8.3. Timeslot allocation problem at SMK

In order to solve the timeslot allocation problem, data from surgeons and patients at SMK is gathered and processed. In particular, the logistic paths of the patients are extracted. These are used in the simulation, and are used to estimate transition probabilities. Furthermore, some parameters are required for the model, such as the available resource capacity and the number of new patients arriving in each time period. This section describes how this data was collected, and which parameters are found from the data to specify an instance of the timeslot allocation problem.

8.3.1. Data collection

The required data was provided by SMK through SAS Enterprise Guide software. As we are dealing with patient data, privacy and anonymity are very important to take into account. It should not be possible to retrace patients from the data. For this reason, the name of the surgeon whose data was used for this instance, and the time period considered, are kept confidential. However, we disclose that data from 2020-2021 is ignored as the COVID-19 pandemic may have distorted healthcare processes.

Since the model describes the waiting list problem for a single surgeon, we only extract logistic patient pathways for patients belonging to the same surgeon. This is because there are differences in transition probabilities between different surgeons. Furthermore, the surgeon chosen works exclusively within one of the SMK healthcare units. This is chosen so that possible differences between patients in different units can be ignored, and are left for future research.

8.3.2. Parameters found from data

8.3.2.1. Available resource capacity

The outpatient department resource capacity is calculated by dividing the number of available OD timeslots in one year, by 26 (the number of two-week periods in one year). This gives $\eta_{OD} = 121$. This should give a good estimate of the average number of OD appointment slots per two weeks. Note that in the real world, the number of OD appointment slots is variable as the number of OD dayparts scheduled varies, as well as the number of assistants assigned to a surgeon during an OD daypart. When using a solution method for real-time planning, the resource capacity can be adjusted per time period.

Similarly, the operating room resource capacity is calculated by dividing the number of surgeries performed by our surgeon in one year, by 26. This gives $\eta_{OR} = 9$.

First consultations take up two OD timeslots, while repeat and discharge consultations only require one timeslot. Surgeries are assumed to take up one OR timeslot. This gives $\zeta_{FC,OD} = 2$, and $\zeta_{RC,OD} = \zeta_{DC,OD} = \zeta_{OR,OR} = 1$.

8.3.2.2. Urgency levels

For first consultations, the maximum waiting time is defined by the Treeknorm at four weeks, so $\mathcal{U}_{FC} = \{2\}$ (recall that one time period is two weeks). For repeat consultations, the desired waiting times are categorized into three urgency levels: six weeks, three months and six months. Therefore, $\mathcal{U}_{RC} = \{3, 6, 12\}$. The OR urgency levels, as discussed in Section 2.2, are $\mathcal{U}_{OR} = \{1, 2, 4, 6\}$. We ignore the acute urgency level as emergency planning is not within the scope of this thesis. For discharge consultations, the standard waiting time is six weeks, so $\mathcal{U}_{DC} = \{3\}$.

8.3.2.3. Transition probabilities

Logistic patient pathways are extracted from SMK data and used to generate vectors of appointment types, for example: $(FC_2, RC_3, RC_6, OR_2, DC_3)$. These are used to generate patients for the simulation, as described in Section 7.2.1. We also use this data to estimate the transition probabilities. These are summarized in Table 8.2. In each row, the probability of a patient transferring to the appointment type indicated in each column is provided. The 'Start' row shows the probability of each appointment type being the patient's first appointment with the surgeon in question. The 'Exit' column shows the probability of the treatment being terminated.

Table 8.2: Transition probabilities for the timeslot allocation problem

	FC_2	RC_3	RC_6	RC_{12}	OR_1	OR_2	OR_4	OR_6	DC_3	Exit
Start	0.7116	0	0.2138	0	0.0185	0.0026	0.0049	0.0264	0.0220	0
FC_2	0.0037	0.2400	0.1298	0.1010	0.0012	0.0049	0.0055	0.0753	0.0147	0.4238
RC_3	0.0028	0.1951	0.1127	0.0919	0.0038	0.0104	0.0189	0.0994	0.0170	0.4479
RC_6	0.0085	0.1575	0.0840	0.0953	0.0028	0.0028	0.0038	0.0660	0.0151	0.5642
RC_{12}	0	0.1535	0.0930	0.0605	0.0023	0.0023	0	0.0628	0.0070	0.6186
OR_1	0	0.2	0.0333	0.0167	0.0667	0	0	0	0.3833	0.3
OR_2	0	0.1	0	0	0.0333	0	0.0333	0.0333	0.6667	0.1333
OR_4	0	0.1522	0.0435	0.0435	0.0217	0	0	0	0.5870	0.1522
OR_6	0	0.1421	0.0299	0.0175	0.0050	0	0	0.0099	0.7182	0.0773
DC_3	0.0021	0.3080	0.2089	0.0654	0	0.0021	0.0021	0.0232	0.0105	0.3776

8.3.2.4. New patients

The number of new patients can be calculated by dividing the strategically budgeted number of new patients that our surgeon should take on in one year by 26. This gives $\lambda_{FC_2} = 28$. However, this is only

an indication of the number of FCs a surgeon should have throughout one year. Table 8.2 shows that it is not uncommon for a patient to start their treatment with an appointment that is not an FC. These are often patients that have been transferred from another surgeon. In our model, we still view these as new patients, since they originate from outside our one-surgeon system. We see that approximately 30% of patients start with a non-FC appointment. Therefore, the total number of new patients should be $28 \cdot \frac{1}{0.7116} \approx 40$.

In the simulation used for LSPI and testing the algorithms, the new patients are drawn randomly from all collected patient pathways, so the expected distribution of first appointments is the same as is indicated in the table. In the LP, we set $\lambda_{j_u} = 40 \cdot q_{Start,j_u}$.

8.3.2.5. Waiting times

In order to be able to create realistic states for the LSPI dataset and for initial states in the simulation, we must find the typical distribution of waiting times of patients on a waiting list for each appointment type at any moment in time. The current waiting times of patients generated for the dataset and simulation should be sampled according to this distribution.

The distribution of waiting times is not very simple. Each urgency level of each queue is differently distributed. The distribution is not only dependent on queue type and urgency level, but also on the length of the waiting list in question: patients will have to wait for a longer time, on average, when the waiting list is long. This introduces variability in the distribution over time. Since determining this distribution is not the focus of this thesis, we choose to use available data on the RC waiting list in order to make a very rough estimate of the distribution, which can be extrapolated to the other queues.

The RC waiting list data consists of the amount of time that patients who are waiting for their RC appointment, have been waiting, measured at some moment in time. We use data for the entire unit of our surgeon, as the data for one surgeon only consists of approximately 250 patients, and may therefore be inaccurate. Using Python, we fit a number of probability distributions (beta, gamma, exponential, truncated normal) to the data. The exponential distribution resulted in the best fit. Figure 8.2 shows a histogram of the distribution of the waiting times of patients on the RC waiting list, and the closest fit exponential distribution, with scale parameter $\beta = 9.015$.

Figure 8.2: Histogram of waiting time distribution for repeat consultations with best-fit exponential distribution

The data shown is for all RC patients, ignoring urgency levels, as these were not available in the dataset used. We expect the scale parameter to be proportional to the respective urgency levels, as patients with a shorter deadline are (in general) treated sooner than those with a longer deadline. Scale

parameter $\beta = 9.015$ was obtained for a combination of urgency levels $3, 6$ and 12. Knowing this, we (very roughly) estimate that the waiting times of patients are exponentially distributed with the scale parameter $\beta = u$, where u is the urgency level of the queue.

Finally, an upper bound must be set on waiting times in order to be able to solve the policy-LP and rolling-horizon LP. As with the large test problem, we set $W_{j_u} = 3 \cdot u$.

8.3.2.6. Costs and rewards

The costs and rewards used in the contribution function should be chosen to reflect the priorities of SMK and relative importance of different appointment types. SMK has indicated that it is especially important to meet the access time norms, described in Section 2.2, for the OR queue. It is worse to have a longer delay for surgeries than consultations, as this has more immediate consequences to a patient's health, and waiting for surgery is, in general, an unpleasant experience. Using all available capacity is also more important for the OR queue, as OR time is much more expensive to waste than OD time.

The hospital has also expressed that meeting urgency level deadlines for repeat consultations is more important than for first consultations. This is because prioritizing patients already in the system allows them to move through their treatment process faster. Also, the article discussed in Section 5.2 [33] showed that prioritizing first consultations just leads to larger backlogs of patients within the system, eventually increasing both internal and external access times. Furthermore, patients waiting to be admitted for a first consultation can still switch to another hospital if their waiting time is too long, which is much more difficult for patients who have already been admitted.

Costs and rewards are determined by repeatedly simulating the system with the rolling-horizon LP as solution method, and adapting costs and rewards until desired behaviour (as described above) is shown. For example, we could see that setting the OR reward too low led to wasted OR timeslots. Resulting from this trial-and-error method, we obtained the cost function $c(j_u, w) = \omega_j \cdot \frac{w}{u}$ for $w \geq u$ and 0 elsewhere, with $\omega_{FC} = 0.5$, $\omega_{RC} = 3$, $\omega_{OR} = 10$ and $\omega_{DC} = 3$. Rewards are set to $r_{FC} = 5$, $r_{RC} = 3$, $r_{OR} = 50$ and $r_{DC} = 3$.

As the policy provided by the solution methods is highly dependent on the costs and rewards in the contribution function, these should be carefully chosen by the hospital. This could be done by repeatedly simulating the solution using different cost and reward settings as we have done, and calculating performance metrics from the simulation, until desired metrics are achieved. An iterative method to update the costs and parameters is proposed by Hulshof et al. in [9] and could also be employed. Alternatively, the costs and rewards could be derived from the actual financial costs associated with wasting resource capacity and increasing waiting times. We do not focus on this aspect of the process as our main goal is to compare the proposed solution methods in terms of the contribution function, so it is sufficient to be consistent in the choice of cost and rewards.

9

Computational results of test problems

This chapter shows the results of various experiments on the small and large test problems. These experiments are aimed at investigating how the performance of the solution methods from Chapter 7 varies as different parameters and aspects of the algorithms are changed, and which settings result in the best performance. We also compare the performance of the solution methods with each other. Throughout this chapter, we assume that we have perfect information about the state for which we wish to select an action, i.e., we do not plan ahead.

When the integer constraint on actions is applied to the rolling-horizon ILP, the program takes approximately ten seconds to solve for the full-size problem. When this constraint is relaxed, the solving time goes down to less than 0.1 seconds, constituting a speed increase of a factor of over 100. As we will be performing many experiments throughout this and the following chapter, in which the program must be solved thousands of times, we choose to relax the integer constraint to a non-negativity constraint during the remainder of this thesis. This does come with some loss of optimality, and therefore we expect results to improve compared to the results shown in this thesis when the integer constraint is applied.

Section 9.1 is devoted to the small test problem, which is used to determine how well the approximate value function found using LSPI can approximate the true value function found using EVI, and which basis functions lead to the best approximation. In Section 9.2, we test different LSPI parameters, explore the effects of changing γ on the rolling-horizon LP, and compare the decision rules introduced in Section 7.4, on the large test problem.

Code used to program solution methods and generate results in this chapter can be found at: `https://github.com/yannavdv/MSc_thesis_Yanna`.

9.1. Small test problem

The small test problem is devised such that it is possible to find the exact solution using exact value iteration. EVI was implemented in Python and used to calculate the exact value function, using $\gamma = 0.9$ and tolerance parameter $\epsilon = 0.1$. After seven iterations, taking over twelve hours, the convergence criterion was met. The resulting value function is used throughout the remainder of this section.

9.1.1. Basis functions

Now the exact value function V^* for the small test problem is known, it can be used to measure which basis functions result in the closest approximation of V^*, using the linear regression method discussed in Section 7.2.3.2. The exact solution provides us with a full dataset of $373,248$ state-value pairs. 10-fold cross-validation was performed on this dataset, comparing each of the proposed basis functions. The 10 subsets are selected randomly. Pairwise combinations of basis functions were also evaluated. In many cases, the combination resulted in the matrix $X^\top X$ being singular, and so no optimal θ^* was

found. These combinations are left out of the results. The average MSEs for the (combinations of) basis functions are shown in Table 9.1.

Table 9.1: Mean squared errors of basis functions for 10-fold cross-validation on test problem

Basis function	**Average MSE**	F
1: Number of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$ who have been waiting for w time periods	2.3055	$(W+1) \cdot U$
2: Number of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$ who are early for treatment, who are on time for treatment, and who are late	2.3054	$3 \cdot U$
3: Cost of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$	4.0474	U
4: Total number of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$	12.0450	U
3 + 4	3.5708	$2 \cdot U$

It is clear that the first two result in the lowest MSE. It should be noted that those functions are very similar for the small test problem. In the queues with urgency level 1, the number of early patients is $s_{j_1,0}$, the number of on time patients is $s_{j_1,1}$ and the number of late patients is $s_{j_1,2}$. The only difference between the basis functions arises for the OR queue with urgency level 0, as it is impossible to be early in that queue, and patients with waiting time 1 and 2 are late. This explains why their MSE's are almost exactly the same.

It is interesting to see that, while basis functions 3 and 4 are of equal dimension, basis function 3 results in a much better approximation of the exact value function. The cost of the patients provides information that is important to be able to approximate the value of the state, as it gives an indication of the number of patients and their combined waiting times. Combining the two functions provides even better results, but is still not as good as the first two basis functions.

It can be instructive to see for which states the difference between the approximated and exact value function is the largest. As the sizes of the states are upper bounded, the value function is likely to be non-linear and therefore more difficult to approximate using a linear parametrisation when the state sizes reach these upper bounds. The absolute error $|V^*(s) - \theta^{*\top}\phi(s)|$, where θ^* is the parameter vector found through linear regression for basis function ϕ, can be compared for different total state sizes $(\sum_{j\in\mathcal{J}} \sum_{u\in\mathcal{U}_j} \sum_{w=0}^{W} s_{j_u,w})$ and for how many of the sub-states have reached their upper bounds. We should also take into account that when there are more states of a certain type in the training set, θ will fit better to such states, resulting in a lower error. Results of these experiments, and plots showing the number of each type of state present in the dataset, are shown in Figure 9.1.

Basis functions 1 and 2 result in almost exactly the same error, which is why basis function 1 seems to be omitted from the results. Figure 9.1(a) shows that the error does increase when the number of upper bound hits increases. However, we also see in Figure 9.1(b) that there are very few states in which many upper bounds are hit, so θ is trained less on such states. Furthermore, Figure 9.1(c) shows a very

large error when the total state size is small. This could suggest that for small state sizes, the approximate value is dominated by the intercept term, as the product $\theta^\top \phi(s)$ without the intercept term is almost always a smaller number when the total state size is smaller. The intercept term fits better to the more average state sizes as these are represented more frequently in the training set.

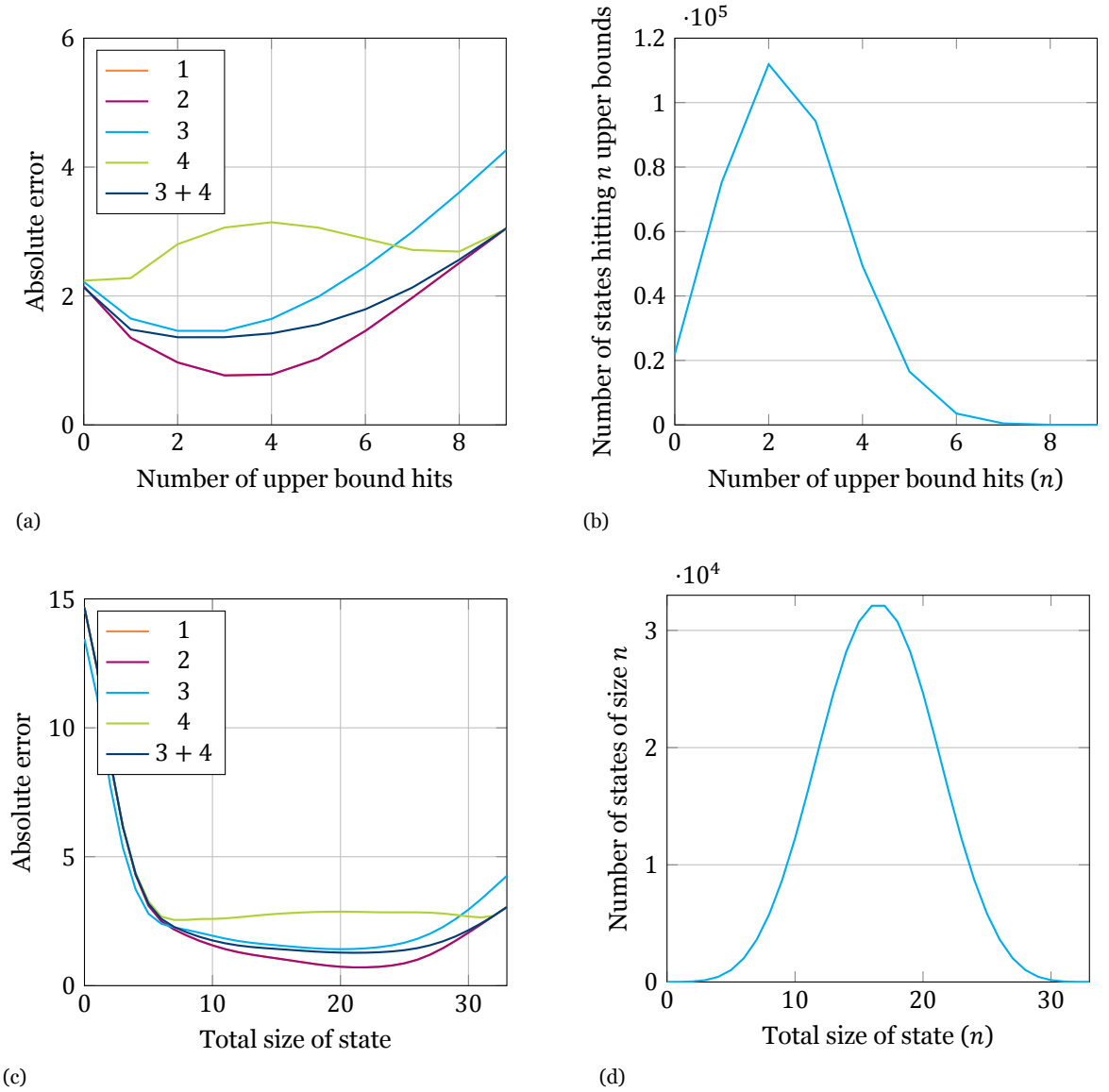

Figure 9.1: Absolute error of different basis functions compared to total size of state and number of upper bounds hit

Finally, we should also consider that the ability of the basis functions to fit the exact solution well does not directly imply that they will result in a good policy. The exact solution is limited by the upper bound on sub-state sizes, which is not the case for the approximate solution. The approximate value function resulting from LSPI may even look quite different from the exact value function, as scaling factors and addition of constant terms do not influence the actions chosen by the resulting policy. For this reason, it is also necessary to investigate the performance of the policies resulting from implementing LSPI using different basis functions, which is done in Section 9.2.

9.1.2. Comparison of solution methods on small test problem

A simulation was created using the parameters and transition probabilities presented in Section 8.1, in order to compare the performance of different policies. The policies which were compared are the approximate policy found through LSPI, the exact policy found through EVI, and the rolling-horizon LP. We use discount factor $\gamma = 0.9$ for all methods. In LSPI, basis function 1 and a dataset size of 5000 are used. We compare the average contribution per time period accumulated by following their respective policies when the number of initial patients is varied. The time horizon is set to 30. 50 trials are performed, and the average contribution per time period accumulated over these trials is calculated. Within each trial, the same simulation is used for each algorithm, meaning that the initial patients and the patients added to the RC queue in each time period are the same.

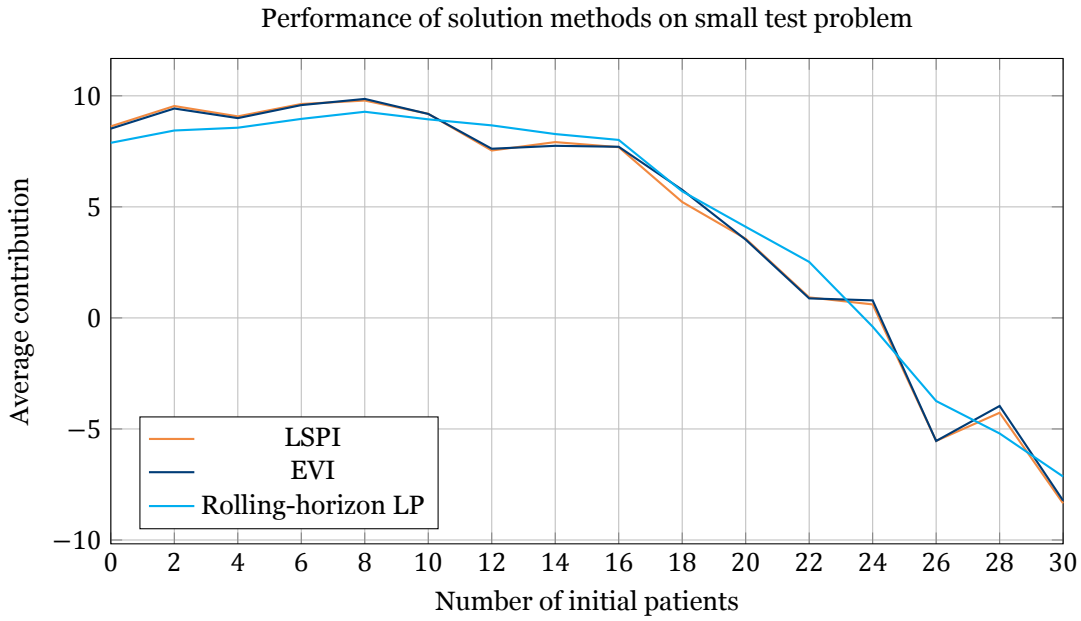

Figure 9.2: Comparison of policies resulting from EVI, LSPI and rolling-horizon LP on the small test problem when the number of initial patients is varied.

Figure 9.2 shows the results of this experiment. Recall that the total contribution should be maximized. We see that the performance of the policies resulting from LSPI and EVI is very similar. This implies that LSPI is able to approximate the exact value function well. However, the rolling-horizon LP produces quite similar results to LSPI and EVI. This is somewhat unexpected, as EVI should give the truly optimal solution, while the LP is only a deterministic approximation of the stochastic MDP. In the next section, we investigate whether we can improve the LSPI algorithm so that it results in a better policy than the rolling-horizon LP.

9.2. Large test problem

The large test problem is constructed as described in Section 8.2. The parameters reflect a scaled-down version of the parameters of the full problem at SMK. Since the problem is scaled down, it is faster to solve, and therefore enables us to perform experiments using the various solution methods within reasonable time. The results of those experiments are described in this section. The static roster method is ignored here, as determining time proportions on the large test problem does not provide us with useful information about the optimal proportions for the full problem.

9.2.1. LSPI parameter tuning

A benefit of LSPI is that it does not have many hyperparameters that require tuning. In the original LSPI algorithm, only γ, ϕ, and M need to be chosen by the user (ignoring the constants ϵ and δ, as these do not have a large influence on the algorithm as long as they are chosen small enough). However, various methods have been proposed in literature, or can be devised by considering characteristics of the problem, to improve the solution quality and/or speed up convergence.

One issue that we should take special care to avoid is that of policy oscillation [35]. This is a phenomenon where LSPI switches, or oscillates, between a number (usually two or three) of different θ's, and therefore never converges. The corresponding policies are usually sub-optimal. Policy oscillation can occur when it is not possible to approximate the true value function using the chosen approximation (convergence is guaranteed when a θ exists such that $V^*(s) = \theta^\top \phi(s)$). The phenomenon is not yet fully understood, difficult to predict, and occurs in many approximate policy iteration algorithms.

Unless stated otherwise, the LSPI algorithm is carried out as shown in Algorithm 6 using basis function 1, discount factor $\gamma = 0.75$, and dataset size $M = 5000$. The same dataset is used for all experiments, with the obvious exception of the experiment on the size of the dataset. Since experiments have shown that the algorithm usually converges within approximately 4 to 6 iterations on the large test problem for similar parameter settings, an upper bound is set on the number of iterations of $N = 20$. If the convergence criterion has not been met by this point, we assume that the algorithm will never converge for the settings used.

The policy resulting from LSPI, which can be described only by the parameter vector θ, is tested on a simulation of the problem for thirty time periods, where the number of initial patients are drawn uniformly from the range $[50, 70]$. This is repeated 50 times. All policies are tested on the same simulations throughout the experiments. The average contribution per time period generated by the policy over all simulations is calculated and is used as the performance measure. The average contribution gained using the rolling-horizon LP on the same simulations is also calculated, to serve as a baseline measurement. Results of all experiments are shown in Figure 9.3. A red cross over a measurement indicates that LSPI did not converge for that parameter setting.

First, the performance of the policy using the four different basis functions is tested; results are shown in Figure 9.3(a). LSPI using basis function 3 (the cost of patients in each urgency queue) does not converge, and results in very poor performance. Despite this, the combination of basis functions 3 and 4 does converge, and results in higher average contribution than basis function 4 alone. LSPI converged for all other basis functions in 4-6 iterations, with basis function 1 resulting in the best performance, and 4 in the worst. For this reason, basis function 1 (essentially the full state vector) is used for LSPI on the full timeslot allocation problem.

The discount factor γ determines to what extent we take into account future costs and rewards. If $\gamma = 0$, the policy is only based on the contribution gained by the current state and action, while for $\gamma \to 1$ we try to look infinitely far into the future. Intuitively, this tells us that increasing γ will improve the policy, but also makes the value function much more difficult to approximate and can result in no convergence. This intuition is confirmed by the experiment testing γ, shown in Figure 9.3(b), where we see that the total contribution increases as γ increases, until LSPI stops converging for $\gamma \geq 0.8$. Choosing $\gamma \approx 0.6$ results in good policies.

Increasing the size of the dataset that θ is trained on, M, should result in better policies as the algorithm can explore a larger part of the state space. The trade-off here is that this will increase the running time of the algorithm (linearly). The results in Figure 9.3(c) show that LSPI will converge reliably for $M \geq 2,500$; the algorithm did not converge for $M = 500$ and $M = 1,000$. Total contribution increases as M increases until $M = 10,000$. Above this, the total contribution starts to decrease. This could be a sign of overfitting, or it could indicate that, above a certain dataset size, the performance of LSPI is more sensitive to the contents of the dataset than to the size. We return to this problem later on.

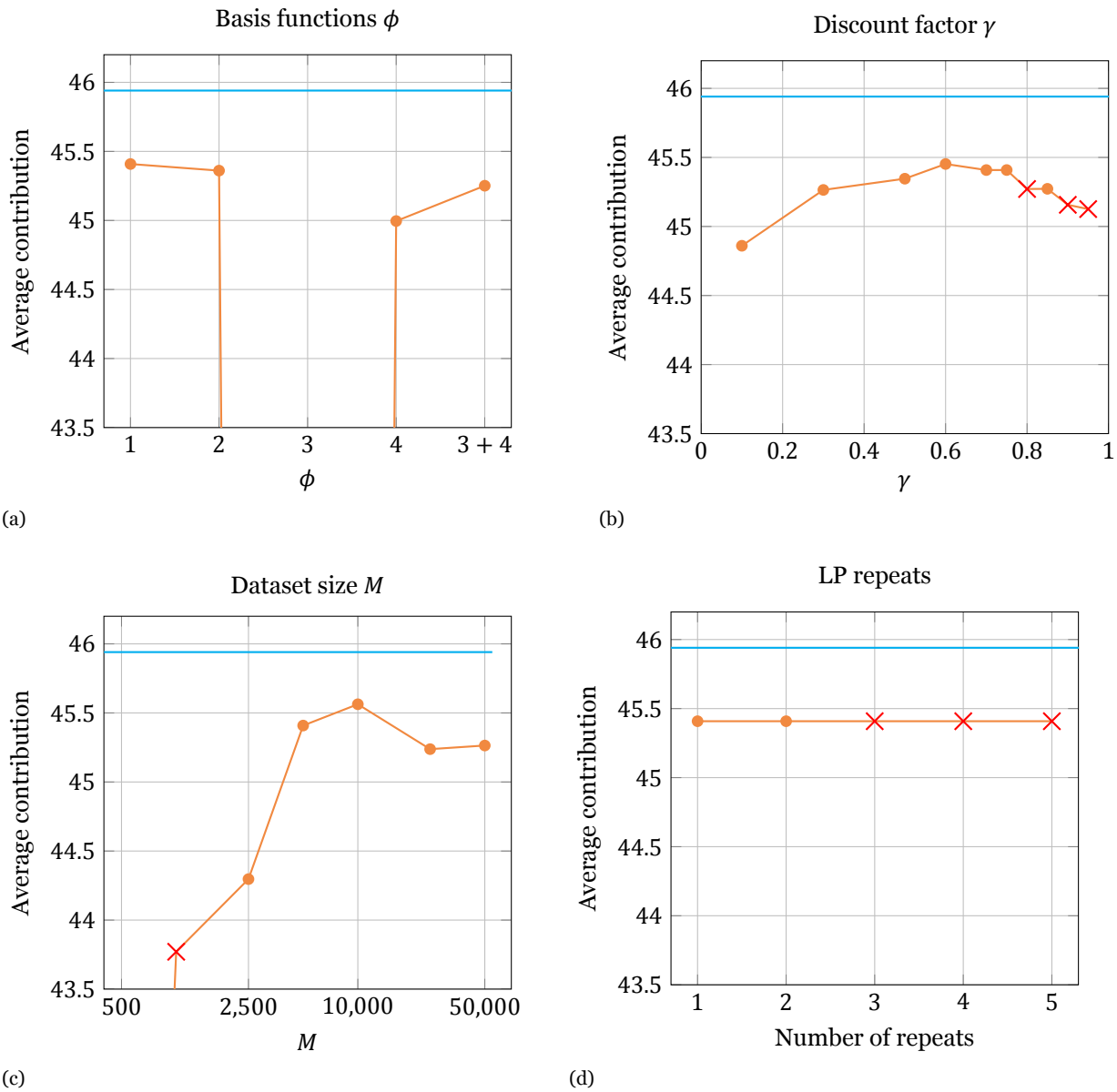

Figure 9.3: Performance of LSPI on large test problem for different parameter settings. Results of LSPI are shown in orange; the baseline measurement of rolling-horizon LP is shown in blue. Settings for which there was no convergence are marked with a red cross.

A drawback to LSPI using a linear parametrisation is that the action selected by the policy is sensitive to small adjustments of θ. This could be a source of policy oscillation. One way to mitigate this could be to repetitively solve the policy-LP with a small amount of Gaussian noise added to θ, and choose the action which occurs most often as the optimal solution, breaking ties by choosing the action resulting in the maximum objective value. A noise vector was added to θ which was normally distributed with mean 0 and variance 0.02, which seemed appropriate in comparison to the size of the components of θ. This was implemented and tested; results are also shown in Figure 9.3(d).

Unfortunately, since solving the policy-LP dominates the running time of the algorithm, doubling the number of repeats also essentially doubles the running time of an iteration. Furthermore, the experiment shows that more repeats actually do not lead to convergence. It is interesting to see that, although the θ's resulting from the runs were slightly different, the results from the simulations were exactly the same, even in case of no convergence. In the unconverged runs, the algorithm oscillated between very similar θ's, the difference being only slightly too large to satisfy the convergence criterion.

This could explain why performance was not affected. We conclude that the LP-repeat method should not be used for this problem.

The following alterations to LSPI were also attempted, but did not result in convergence:

- Off-policy LSPI-Q was implemented. Since this algorithm learns from state-action pairs, for every state that would be used for LSPI-V, multiple actions should be sampled. Since the action space for this problem is very large, this requires a much larger dataset. Any reasonable increase in size of the dataset did not result in convergence.

- On-policy LSPI-V was also implemented. Since the states in an on-policy trajectory are likely to be very similar, in order to sufficiently explore the state space, it is necessary to implement more, but shorter, iterations (i.e., increase N and decrease M). If N is large and M is small, θ should be updated after a number of iterations rather than every iteration, so enough data is seen before each update. This unfortunately did not lead to convergence on the large test problem.

- Replacing a (small) percentage of the dataset in each iteration by random states, or successor states from the previous iteration, also did not work. This observation, combined with the previous point, suggest that the dataset should not be changed when running LSPI.

- Rather than using θ^{n-1} for the policy in iteration n, we can use a weighted combination of θ's calculated in previous iterations, with a higher weight assigned to those from most recent iterations. The idea behind this is that it will cause smaller jumps between value function approximations, and so smaller jumps between successive policies, possibly damping any policy oscillation. This was implemented and tested, but did not have the desired effect. This could be because a weighted combination of good value function approximations does not necessarily result in a new good policy.

- In LSPI, the policy always chooses the best action based on the information at hand. This could cause the policy to get stuck in a local optimum. A popular method to escape from local optima and enforce more exploration is to use an ϵ-greedy policy [36], which chooses the best action with probability $1 - \epsilon$, and a random action with probability ϵ. This was implemented and tested for different values of $\epsilon \in (0, 0.2]$, but unfortunately this, too, increased policy oscillation.

- In their paper introducing LSPI, Lagoudakis & Parr indicate that the sample states in the dataset should be similar to the states that will be visited by the policy in real life [11]. This is somewhat paradoxical, as we need the dataset to find the policy, but need the policy to define a good dataset. We can solve this problem by first running LSPI on a randomly generated dataset, and using the resulting policy in simulations to collect new states. A portion of the first dataset is replaced by these new states. Although this did not affect convergence, the performance did get worse, even for relatively small replacements of the dataset. This could be because the states encountered in the simulation are usually quite good, as they are the result of an already quite good policy. It is possible that the policy benefits more from learning to stay away from bad states, rather than learning to go towards good states. This idea should be investigated further.

9.2.2. Influence of discount factor on rolling-horizon LP

The discount factor, γ, is not only used in LSPI, but also in the objective function of the rolling-horizon LP. Since we are using the rolling-horizon approach, it could be beneficial to reduce γ to give relatively more weight to time periods in the near future, as due to uncertainty, the states in the far future will be less similar to the expected states in the LP. Furthermore, it is interesting to see which effect decreasing γ has on the time required to solve the LP.

An experiment was done measuring the average contribution per time period and running time of the rolling-horizon LP on a simulation of the large test problem, where the average is taken over 50 trials. The time horizon and number of initial patients are both set to 30. Results are shown in Figure 9.4.

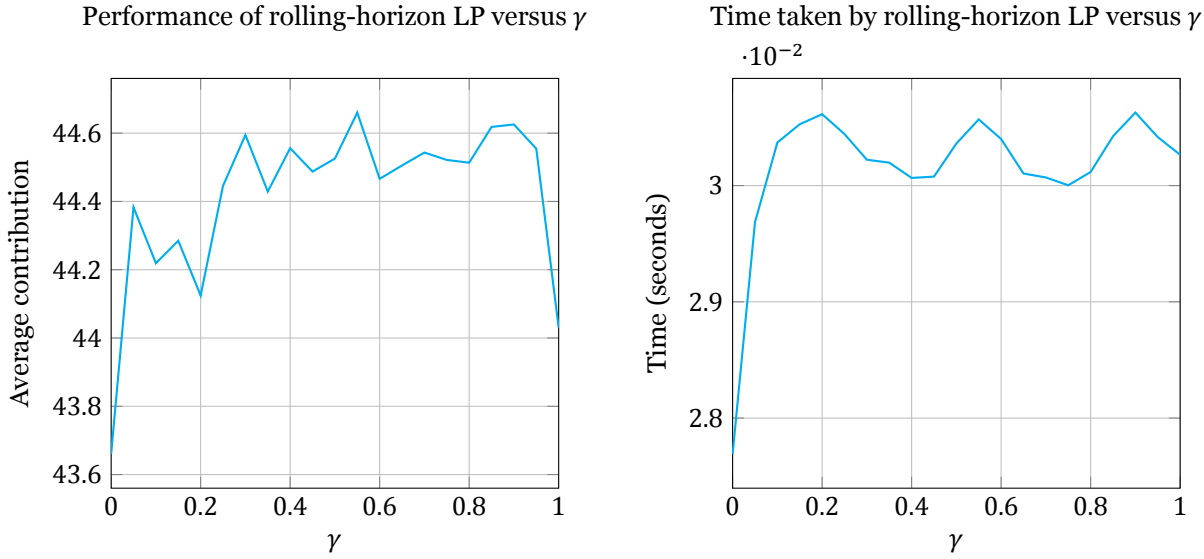

Figure 9.4: Absolute error of different basis functions compared to total size of state

Figure 9.4 shows that initially, contribution and time taken increase as γ increases. Although there are minor fluctuations, the region $\gamma \in [0.3, 0.95]$ appears to reliably result in good performance. Interestingly, there is a large drop in performance from $\gamma = 0.95$ to 1. The discount factor does not appear to have much influence on running time for $\gamma \geq 0.1$.

9.2.3. Evaluation of decision rules

The four decision rules described in Section 7.4 were implemented and tested on the large test problem. Figure 9.5 shows the average contribution per time period gained when following each of the decision rules in the simulation for thirty time periods. The average is calculated over 50 trials.

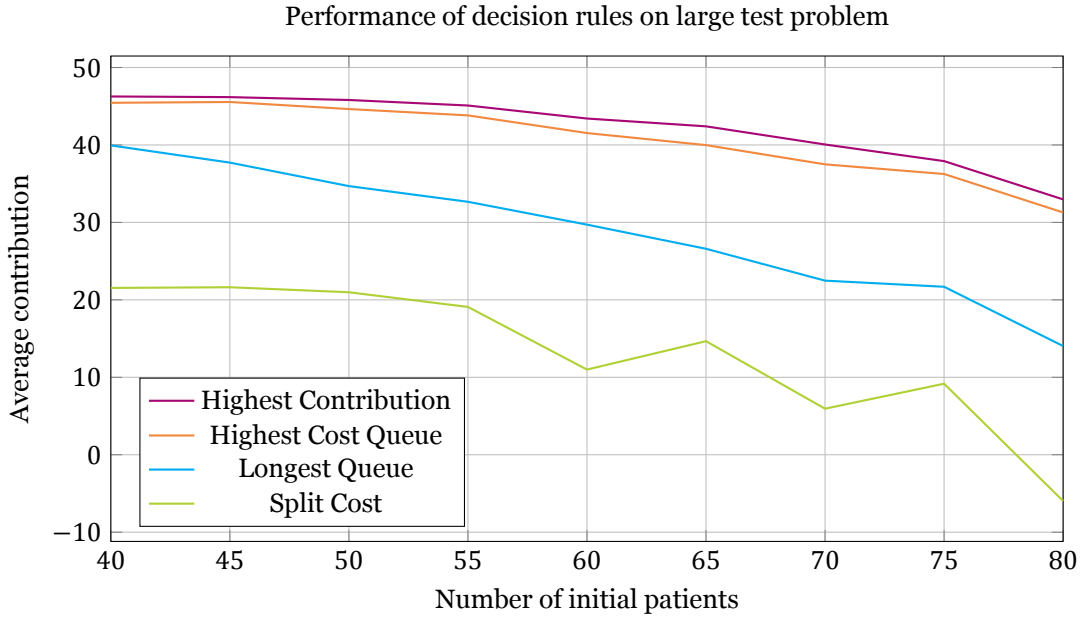

Figure 9.5: Comparison of the decision rules introduced in Section 7.4. The time horizon is set to 30 time periods.

The results show that the Highest Contribution rule performs best, followed closely by the Highest

Cost Queue rule. The Split Cost rule performs worst. The Highest Contribution rule is a direct, greedy maximization of the contribution function within each time period, without taking future effects of the decisions into account. Technically, the rule is equivalent to the rolling-horizon LP with $\gamma = 0$, as in that case the action is selected which maximizes the contribution in only the current time period. Because of this, we expect the performance of the decision rule to be slightly worse than that of the rolling-horizon LP.

9.2.4. Comparison of solution methods on large test problem

We can now apply what has been learned from the experiments described in this chapter to each of the solution methods, and compare their performance in order to determine which results in the best policy. The following settings are used:

- **LSPI:** Basis function 1 is used, γ is set to 0.6, the dataset size M is $10,000$, and no LP repeats are done.
- **Rolling-horizon LP:** We set $\gamma = 0.5$, as this region reliably resulted in high contribution.
- **Decision rule:** The Highest Contribution rule is used, as this consistently resulted in the highest contribution out of all decision rules tested.

We compare the total contribution accumulated by following each of these solution methods in a simulation, when the number of initial patients is varied. The time horizon is set to 30 time periods. Each trial is repeated 50 times, and the average contribution per time period over the trials is calculated. The results are shown in Figure 9.6.

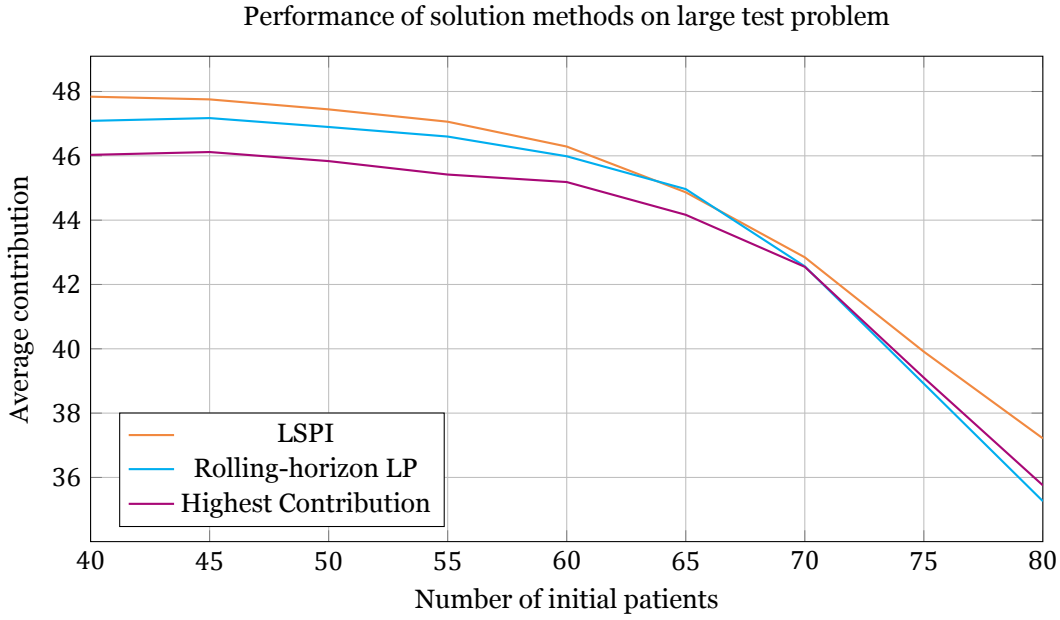

Figure 9.6: Comparison of policies resulting from LSPI, the rolling-horizon LP and the Highest Contribution decision rule on the large test problem when the number of initial patients is varied.

The results show that LSPI almost always results in a higher average contribution than the rolling-horizon LP and the decision rule. Adjusting both the dataset size and γ was successful in improving the performance of LSPI such that it now outperforms the LP. However, the difference in performance between the solution methods is relatively small, and to a hospital, it is worth considering that the LP and decision rule are much faster, more reliable and easier to implement than LSPI.

All solution methods result in lower contribution when the number of patients is increased; this is a result of more patients being too late as the waiting lists are already too long. The maximum possible reward is already achieved when all resource capacity is maximally used. It is interesting to see that

the total contribution is relatively stable until it starts to decline sharply after 60 patients. This could indicate a tipping point of some sort in the system, after which it gets increasingly difficult to attend to the backlog of patients.

Finally, we should note that it is interesting that setting γ as low as 0.6 for LSPI, or 0.5 for the LP, actually improves performance in comparison to higher values of γ. This implies that our expectation of the impact of our decisions on the future should be heavily discounted: the contribution four time periods (two months) from now is worth one-sixteenth of the current contribution (for $\gamma = 0.5$). A reason for this could be that the outcomes of the chosen actions can differ so much from the expected outcome, that it is not worth trying to take these into account, and it is better to focus on maximizing the immediate contribution. This hypothesis is supported by the fact that the decision rule performs so well, even though the rule cannot take any future effects into account. Even stronger: the Highest Contribution rule is equivalent to the rolling-horizon LP with $\gamma = 0$, as it directly maximizes the contribution in the current time period under the system constraints.

10

Results of case study

Now that we have determined which parameters to use for LSPI and the rolling-horizon LP, and which decision rule is best, we can apply the solution methods to the full timeslot allocation problem at SMK. This will allow us to make a recommendation on whether the hospital should adopt one of the new methods developed in this thesis for the timeslot allocation phase of the planning process. The results shown in this chapter use the data presented in Section 8.3.

We compare the performance of the solution methods from Chapter 7 in Section 10.1. This comparison still assumes we have perfect information about the state for which we wish to select an action. How well the solution methods work when this assumption is released, and we plan multiple time periods ahead, is assessed in Section 10.2. Here, we investigate how much our prediction of the future state differs from reality, and whether prediction changes the influence of the discount factor in the rolling-horizon LP. A comparison of the solution methods when we increase the planning horizon is done. Using the knowledge gained from these results, a hybrid method is proposed in Section 10.3. Finally, we compare how well solution methods adhere to internal and external access time norms in Section 10.4. Code used to program solution methods and generate results in this chapter can be found at: `https://github.com/yannavdv/MSc_thesis_Yanna`.

10.1. Comparison of algorithms

We first compare the performance of the policies resulting from LSPI, the rolling-horizon LP, the Highest Contribution decision rule and the static roster used by SMK. Here, we assume that we have perfect information about the state for which the policy must choose an action, i.e., we do not plan ahead. As with the test problems, trials were performed in a simulation and the average contribution per time period is measured when the number of initial patients (i.e., the number of patients in the simulation at $t = 0$) is varied. The time horizon is set to one year (26 time periods). The average is calculated over 100 trials; the number of trials is increased from the previous chapter due to the relatively higher importance of these results. Results are shown in Figure 10.1. The following settings are used for the solution methods:

- **LSPI:** Basis function 1 is used, γ is set to 0.6, the dataset size M is $50,000$, and no LP repeats are done. Due to the increased size of the problem, we expect a larger dataset than in the large test problem is necessary to achieve good results. The LSPI algorithm took 3 iterations, lasting 4 hours, to converge to a parameter vector.
- **Rolling-horizon LP:** We set $\gamma = 0.75$, as in Section 10.2.2 we show that this reliably results in better performance than lower values.
- **Decision rule:** The Highest Contribution rule is used.
- **Static roster:** The data suggests that approximately 50% of OD time should go to FC appointments, 42.5% to RC appointments and 7.5% to DC appointments. This translates to $a_{FC,t} =$

30, $a_{RC,t} = 52$, and $a_{DC,t} = 9\ \forall t \in \mathcal{T}$ (recall that FC's take twice as long as RC and DC appointments). All OR capacity should be used, if possible. Within each queue, appointments are allocated to patients with the highest cost.

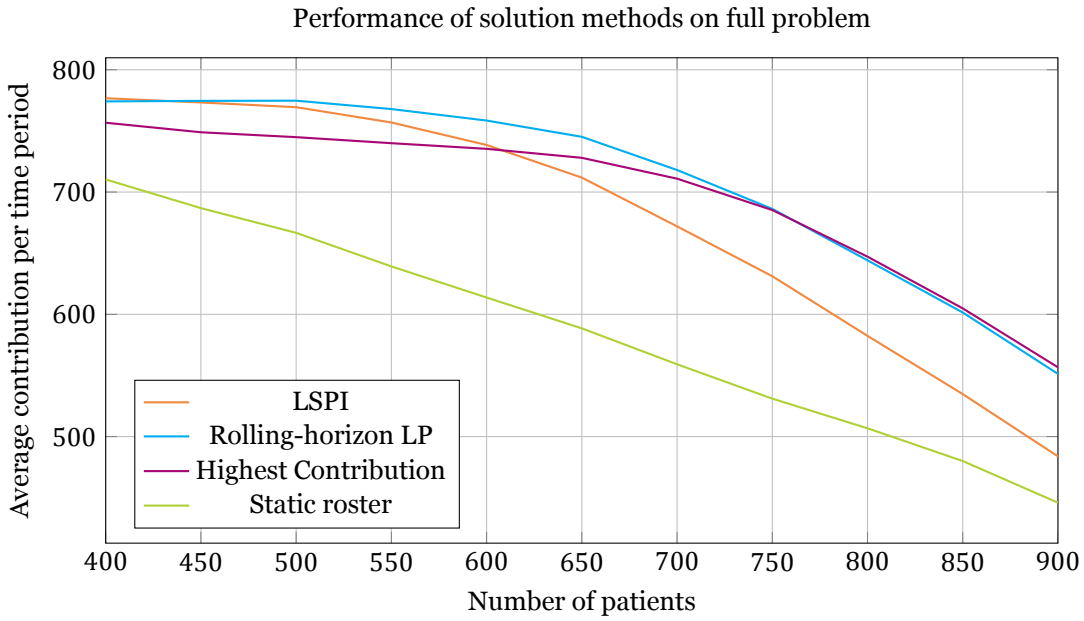

Figure 10.1: Comparison of policies resulting from LSPI, the rolling-horizon LP, the Highest Contribution decision rule and static rosters on the full problem when the number of initial patients is varied.

The results show that for 400 initial patients, LSPI results in the highest contribution, although only marginally higher than the LP. The performance of LSPI quickly drops to a level below the LP and decision rule as the number of patients initially in the system is increased. The rolling-horizon LP steadily achieves good results, although it is slightly overtaken by the decision rule for over 750 patients. The lowest contribution is accumulated by the static roster method of SMK.

It is interesting to see that the relative performance of the solution methods differs from what we saw for the large test problem in Section 9.2.4, where LSPI steadily performed best. It is possible that the increased size and complexity of the problem cannot be handled well by a policy based on a linear parametrisation. However, increasing the complexity of the policy, for example using non-linear functions, will make the algorithm much slower and prohibits the use of the policy-LP.

The fact that the Highest Contribution decision rule performs slightly better than the rolling-horizon LP for a high number of patients is also interesting, as the decision rule is essentially the same as the LP when we set $\gamma = 0$. This would suggest that with a higher number of patients, it is better to decrease γ in order to deal with the high waiting lists immediately. This could be exploited through a scheme where γ is adapted depending on the expected number of patients.

10.2. Planning ahead

We have seen that the solution methods investigated provide policies which appear to manage the waiting lists better than the current method used by SMK. However, until this point we have assumed that decisions can be made with perfect information about the current state of the system. We now investigate what happens in the more realistic situation that timeslots must be allocated to appointment types several weeks in advance, and a prediction of the future state must be used to make this scheduling decision, as explained in Section 6.4. Since our prediction of the future state will differ from the actual state, we expect to lose some performance quality. In this section, we investigate how much is lost, and if there is a difference in the relative performance of the algorithms, when we plan ahead.

10.2.1. Predicting the future state

We first measure how far the predicted state, calculated using Algorithm 3, differs from the realization of that state in the simulation. The predicted state at time t is denoted by $\tilde{s}_t$. The difference when predicting P time periods ahead can be calculated on three "levels":

1. The exact difference between the predicted and true state vectors:

$$\sum_{j\in\mathcal{J}}\sum_{u\in\mathcal{U}_j}\sum_{w=0}^{W}\left|s_{j_u,w,t+P}-\tilde{s}_{j_u,w,t+P}\right|.$$

2. The difference between the number of patients in each urgency level of each queue (ignoring waiting times):

$$\sum_{j\in\mathcal{J}}\sum_{u\in\mathcal{U}_j}\left|\sum_{w=0}^{W}s_{j_u,w,t+P}-\tilde{s}_{j_u,w,t+P}\right|.$$

3. The difference between the number of patients in each queue (ignoring urgency levels and waiting times):

$$\sum_{j\in\mathcal{J}}\left|\sum_{u\in\mathcal{U}_j}\sum_{w=0}^{W}s_{j_u,w,t+P}-\tilde{s}_{j_u,w,t+P}\right|.$$

The difference between true and predicted state will be largest at level 1 and smallest at level 3. Furthermore, we expect the difference between the true and predicted state to grow as the number of time periods to plan ahead, P, increases.

Figure 10.2 shows how much the true and predicted states differ from each other, as a fraction of the total number of patients in the true state. This is done by performing P static roster actions, starting from a random state s_t, in a simulation to determine the true state s_{t+P}. Those actions, and the state s_t, are then used to compute the predicted state $\bar{s}_{t+P}$. 1000 trials are performed; the average difference over these trials is shown.

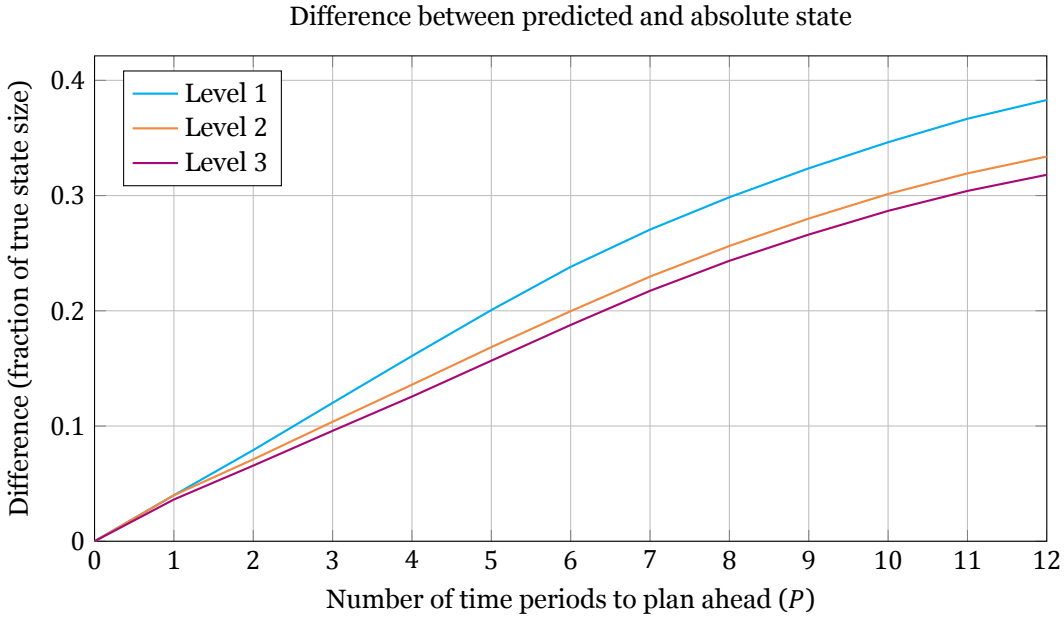

Figure 10.2: The difference between the predicted state, determined using Algorithm 3, and the realized state within the simulation when planning P time periods ahead, as a fraction of the realized state.

The results show that the prediction becomes worse as the number of time periods we want to plan ahead increases. After $6-8$ time periods, the rate of change decreases somewhat. SMK plans twelve weeks, or six time periods, ahead; at that point, we see that on average, approximately 20% of patients are in different queues than we expect them to be. It is likely that this difference negatively impacts performance of the solution methods. A large degradation in performance could provide motivation for SMK to reduce the planning horizon for OD appointments. This is investigated in Section 10.2.3.

10.2.2. Discount factor for rolling-horizon LP

In Section 9.2.2, we saw that increasing the discount factor γ initially improved performance of the rolling-horizon LP, but that performance remained relatively stable in the range $\gamma \in [0.3, 0.95]$. In this section, we investigate whether this is also the case for the full problem, and if the influence of the discount factor is different when we plan ahead.

An experiment was done measuring the average contribution per time period and running time of the rolling-horizon LP on the full timeslot allocation problem when the discount factor is varied. The average is taken over 50 trials. We compare the performance without planning ahead ($P = 0$), and when planning 6 weeks ahead ($P = 6$). The time horizon is set to one year (26 time periods) and the number of initial patients is drawn from $\mathcal{N}(700, 200)$ in each trial. The running time is measured as the amount of time it takes to solve the LP once. Results are shown in Figure 10.3.

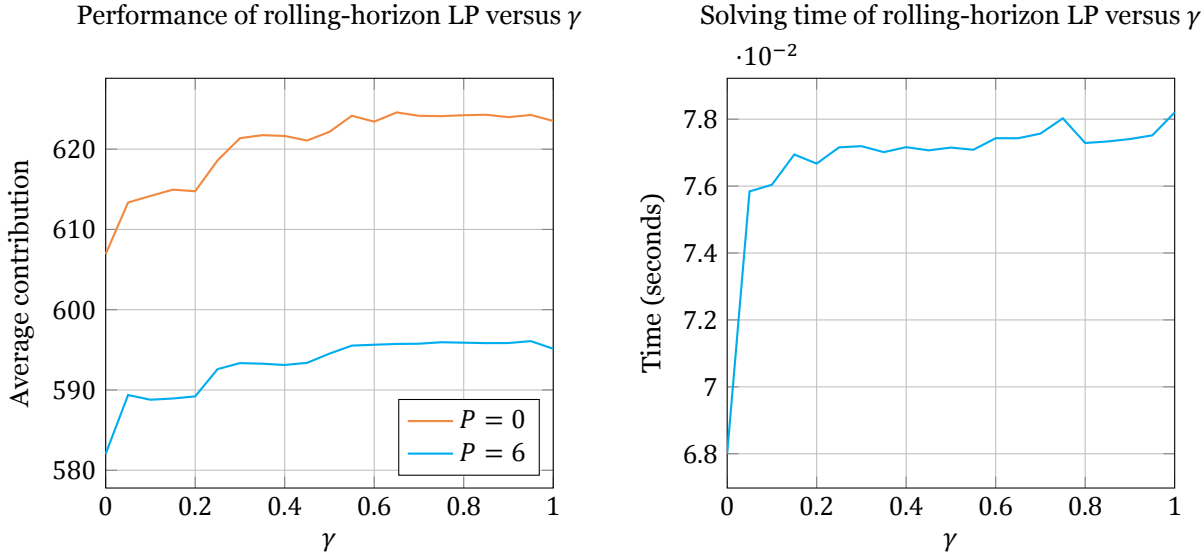

Figure 10.3: Influence of γ on rolling-horizon LP performance and time

Now, we see a clear positive trend when γ is increased. The trend is similar for the true and predicted case. Most notable are the jump increase in both contribution and time taken from $\gamma = 0$ to $\gamma = 0.05$, and the flattening of the trend from $\gamma = 0.7$ onwards. There is a large increase in time taken to solve the LP from $\gamma = 0$ to $\gamma = 0.05$; after this, the time taken increases only slightly as γ increases. We use $\gamma = 0.75$ in the experiments in this chapter to achieve best performance of the LP.

10.2.3. Performance when predictions are used

We expect the performance of all solution methods to be worse when the predicted state is used rather than the true state, as we have seen that these generally differ quite a lot from each other, and so our predictions about the value of actions can also be far from the truth. The only method that will not suffer from this is the static roster method, as this will always provide the same action, regardless of the state.

Figure 10.4 shows a comparison of the total contribution accumulated in a simulation by the dif-

ferent solution methods. The average contribution is calculated over 100 trials. The same parameter settings are used for the solution methods as in Section 10.1. The number of initial patients in the simulation is drawn in each trial from $\mathcal{N}(700, 200)$, and the time horizon is one year (26 time periods). When planning P weeks ahead, in each time period, we calculate the predicted state for the current time period, $\tilde{s}_t$, using the true state P time periods ago, s_{t-P}, and the actions performed in the meantime. The first $P+1$ time periods are ignored in the total contribution, as this is a warmup-period where s_{t-P} does not yet exist.

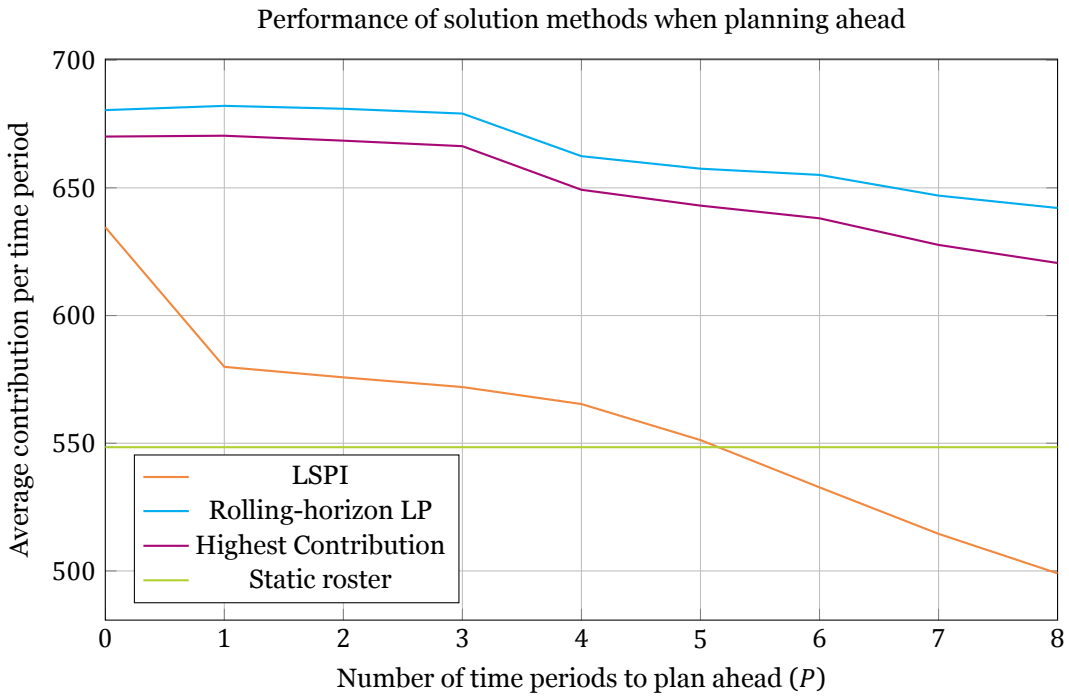

Figure 10.4: Comparison of policies resulting from LSPI, the rolling-horizon LP, the Highest Contribution decision rule and static rosters on the full problem when planning P time periods ahead.

We see that, as expected, performance of the rolling-horizon LP, LSPI, and the decision rule decrease as the planning horizon increases, while that of the static roster remains the same. Especially the performance drop of LSPI from $P = 0$ to $P = 1$ is very dramatic, and eventually, the algorithm is even worse than the static roster method. It is interesting, then, that the rolling-horizon LP works so much better, while it is designed to solve the same problem as LSPI. This suggests that the linear parametrisation used in LSPI is too much of a simplification of the complex real-world problem. The performance of the LP and Highest Contribution decision rule does not decrease much from $P = 0$ to $P = 3$, indicating that planning 6 weeks in advance will not negatively impact waiting times or capacity use. This could be because our prediction of patients with high waiting times is still quite good at this point; the main difference in the true and predicted states will lie with recently treated patients, who will still have short waiting times. Finally, we also see that the decision rule works quite well consistently, although it is not as good as the LP.

10.3. Hybrid method

The results above show that the contribution accumulated by the solution method decreases as the planning horizon increases. In the real world, this implies that patients must wait longer for treatment, or capacity is not fully used. This provides motivation for decreasing the planning horizon for this phase of the planning process. However, this also negatively impacts patients as this delays the time from which their appointments can be booked. Therefore, we propose a hybrid method which hopefully achieves the best of both worlds.

The hybrid method combines the static roster method with the rolling-horizon LP. Although it would be bad to shorten the planning horizon for all appointments, we could allow for a long planning horizon (of 12 weeks, as is the case now) for a portion of the appointments, and use a shorter planning horizon for the remaining appointments. This allows us to make many appointments far enough in advance, while exploiting the improved knowledge we have of the predicted state when planning less far ahead. The static roster method is used to determine the allocation of α% of the timeslots 6 time periods (12 weeks) before the time period to be scheduled. This is the *fixed* schedule portion. Then, we use the rolling-horizon LP τ time periods prior to the week to be scheduled, to determine the allocation of the other timeslots, making up the *dynamic* schedule portion.

Let $\overline{a}_j$ be the number of patients to treat from queue j, as determined by the static roster. Let K be some large number. Recall that $s_{j_u,w,t}$ denotes the sub-state size and $a_{j_u,w,t}$ the number of patients to treat from that sub-state. The following inequalities should be added to the rolling-horizon LP to find the dynamic portion of the schedule:

$$\overline{a}_j \cdot \frac{\alpha}{100} - \sum_{u\in\mathcal{U}_j}\sum_{w\geq 0} s_{j_u,w,t} \leq K \cdot y_{j,t}, \qquad \forall j \in \mathcal{J} \quad \forall t \in \mathcal{T},$$

$$\sum_{u\in\mathcal{U}_j}\sum_{w\geq 0} s_{j_u,w,t} - \overline{a}_j \cdot \frac{\alpha}{100} \leq K \cdot (1 - y_{j,t}), \qquad \forall j \in \mathcal{J} \quad \forall t \in \mathcal{T},$$

$$\sum_{u\in\mathcal{U}_j}\sum_{w\geq 0} a_{j_u,w,t} \geq \sum_{u\in\mathcal{U}_j}\sum_{w\geq 0} s_{j_u,w,t} - K \cdot (1 - y_{j,t}), \qquad \forall j \in \mathcal{J} \quad \forall t \in \mathcal{T}$$

$$\sum_{u\in\mathcal{U}_j}\sum_{w\geq 0} a_{j_u,w,t} \geq \overline{a}_j \cdot \frac{\alpha}{100} - K \cdot y_{j,t} \qquad \forall j \in \mathcal{J} \quad \forall t \in \mathcal{T},$$

$$y_{j,t} \in \{0,1\}, \qquad \forall j \in \mathcal{J} \quad \forall t \in \mathcal{T}.$$

The first two inequalities determine which value is smaller: $\sum_{u\in\mathcal{U}_j}\sum_{w\geq 0} s_{j_u,w,t}$, the number of patients in queue j at time t, or $\overline{a}_j \cdot \frac{\alpha}{100}$, the action already fixed by the static part of the roster. The second two inequalities ensure that $\sum_{u\in\mathcal{U}_j}\sum_{w\geq 0} a_{j_u,w,t}$, the action chosen for that queue, is at least as large as the minimum of those two values. This is necessary because simply forcing the action to be larger than the static part of the roster can lead to infeasibility, if the size of the queue is smaller in that time period.

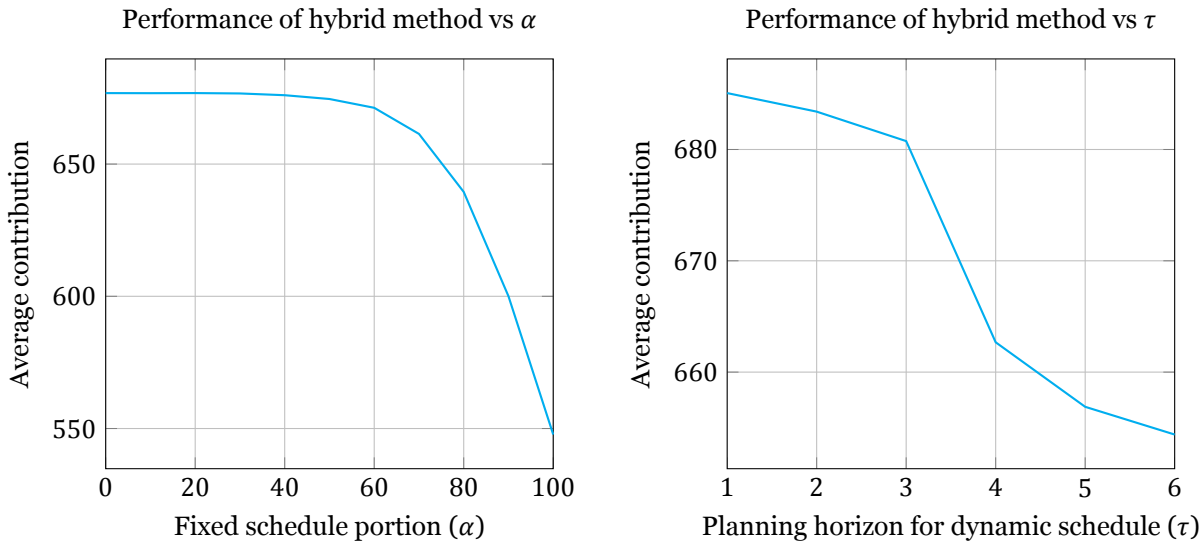

Figure 10.5: Influence of α and τ on performance of the hybrid method

We show the average contribution per time period accumulated over one year from applying this

method for different values of α and τ in Figure 10.5. The average is taken over 50 trials. When α is varied, τ is fixed to 3 time periods. When τ is varied, α is fixed to 50%. Ideally, we want to choose α and τ as high as possible without significant performance loss, as this means that more appointments can be booked further in advance. We see that the average contribution remains steadily high until $\alpha = 60\%$, then quickly drops to its lowest point at $\alpha = 100\%$ (a fully static roster). For τ, there is a large jump in performance loss from $\tau = 3$ to $\tau = 4$. Following this, we would recommend fixing 60% of the OD appointments using the static roster method, and allocating the remaining 40% of timeslots 3 time periods (6 weeks) in advance using the rolling-horizon LP.

10.4. Access times

While it is clear from the results shown in this chapter that the solution methods discussed here improve on the static roster method used by SMK in terms of the contribution function, we should also compare how well the methods adhere to the internal and external access time limits. In Section 2.2, we showed statistics on this from SMK data. However, it would be unfair to directly compare results from the simulation with real-world results, as the simulation does not take cancellations, no-shows or fluctuations in the number of new patients, weekly capacity, or other real-world imperfections into account. Therefore, we use the static roster method as the SMK baseline measurement, and compare it to the hybrid method with $\alpha = 60\%$, $\tau = 3$, and the rolling-horizon LP with planning horizon 6 (twelve weeks). The percentage of patients meeting their urgency level deadline and average access time (in time periods) for these methods, where the average is taken over 100 trials in the simulation, are shown in Table 10.1.

Table 10.1: Access time statistics achieved by hybrid method ($\tau = 3, \alpha = 60\%$), rolling-horizon LP ($P = 6$), and static roster.

Queue	Appointments within time limit (%)			Average access time (time periods)		
	Hybrid	LP	Static	Hybrid	LP	Static
FC_2	26.87	26.02	3.27	3.83	3.83	4.80
RC_3	99.97	93.01	99.77	2.78	2.84	1.19
RC_6	100	100	99.87	5.20	5.11	0.67
RC_{12}	100	100	99.96	7.62	4.34	0.23
OR_1	96.76	96.09	96.52	0.97	0.97	0.99
OR_2	97.66	97.17	97.05	1.85	1.86	1.89
OR_4	97.15	96.64	97.46	3.47	3.51	3.52
OR_6	98.60	98.16	97.97	1.86	1.94	2.04
DC_3	100	82.24	95.34	2.46	2.53	0.84

For most appointment types, the hybrid method marginally increases the percentage of appointments that are done within the urgency level deadline, compared to the rolling-horizon LP and static rosters. The exception is first consultations, where an improvement from 3.27% to 26.87% is realized. There is also a 5% increase in the number of on-time discharge consultations from static rosters to the hybrid method.

It is interesting to see that, while the average access times for repeat and discharge consultations are much shorter when using the static roster, the hybrid method achieves a higher percentage of appointments within the time limit, as does the rolling-horizon LP for RC's. This is because in the contribution function used in the LP, no cost is assigned for making a patient wait for another time period when they are still within their deadline. The LP also has more flexibility in prioritizing late FC or DC patients over RC patients when necessary, and is better at handling unexpected outlier patients. This presents a trade-off to hospital management: ensure that on average, patients move through the treatment process quickly, or ensure that as many patients as possible are treated within their deadline? It

is not immediately obvious that these goals are in conflict, yet these results show that this is the case. Sometimes, less urgent patients must wait for a relatively longer time in order to ensure that more urgent patients can be treated on time.

Another goal was to make maximal use of the available resource capacity. In the same experiment as above, we also measured how much OD and OR capacity was left unused by both methods during one year. The hybrid method left 0.76% of OD capacity and 0.31% of OR capacity unused on average. The rolling-horizon LP wasted 1.15% of OD capacity and 0.35% of OR capacity. The static roster wasted 2.70% of OD capacity, but only 0.18% of OR capacity. However, during this experiment (as with all other experiments in this thesis), the integer constraint on actions in the LP was relaxed to a non-negativity constraint. Otherwise, the experiment would have taken around twenty hours to run, as opposed to the fifteen minutes it took with the relaxation. This results in some wasted capacity, as the actions selected by the LP must be rounded down to the nearest integer to ensure feasibility. We expect less wasted capacity, and even better access times, when the integer constraint is applied.

We should point out that these results are very dependent on the cost and reward parameters, in the case of the LP and hybrid method, and on the proportions used in the static roster method. For example, the results shown in Table 10.1 suggest that the proportion of first consultations in the static roster was probably too low; however, increasing this proportion will have a negative impact on RC and DC access times. As for costs and rewards, assigning a higher cost for making FC patients wait will likely lead to shorter external access times but longer internal access times. Of course, it is up to the hospital to decide how to prioritize each appointment type.

11

Conclusions and recommendations

The aim of this thesis was to answer the research question:

How can timeslot allocation of consultation types be used to control waiting times of patients and use of resource capacity?

The planning intervention studied in this thesis was determining an allocation of timeslots to consultation types in schedules of surgeons, with as a case study the orthopaedic department of the Sint Maartenskliniek in Nijmegen. We saw that the hospital had difficulty in meeting Dutch and self-imposed norms on access times; our goal is to improve on this so that patients have to wait less long for treatment, positively impacting their health and satisfaction. Furthermore, a requirement was to make maximal use of the available resource capacity (i.e., OD and OR time), in order to treat as many patients as possible, and keep idle time to a minimum.

In this chapter, we discuss how this problem was modelled, which solution methods were attempted, and what we can conclude from their results in Section 11.1. Our recommendations for future research and for implementation of our results are given in Section 11.2.

11.1. Conclusion

The timeslot allocation problem can be classified as a sequential decision problem under uncertainty, and was modelled as a Markov decision process. The model was based on the general model provided by Hulshof et al. in [10]. We added urgency levels within patient queues to represent the deadline before which a patient must be treated. The objective is to minimize patient waiting times and maximize the number of patients treated. This was translated to a contribution function by assigning costs for making patients wait longer than they should, and rewards for treating patients. We also considered that appointment schedules must be generated well in advance, and developed a method to predict the future state for which planning decisions must be made.

Multiple solution methods were explored. The first approach considered was to find the truly optimal policy using exact value iteration. However, the size of the state space in any real-world realization of the problem means that no exact solution can be found within reasonable time. Therefore, a form of approximate dynamic programming must be used. Least-squares policy iteration was chosen as it can handle infinite time horizon problems and has shown better experimental results than other methods. In literature, both an off-policy version in which the policy is defined by a Q-function [11], and an on-policy version in which the policy is defined by a value function [12], exist. We implemented an off-policy version in which the policy is defined by a value function, as the action space is too large to be able to use a Q-function, and the on-policy variant did not converge for our problem. The policy uses an (I)LP to choose a feasible action maximizing the expected immediate contribution and value of future states following from that action. For this reason, we are constrained to using linear basis functions to represent the value function. Having linear basis functions also allows us to use the expected next state,

rather than the post-decision state used in [12] and [10], to calculate the expected value of the next state.

Next to these methods, we also investigated options that could be easier for a hospital to implement. The first was an (integer) linear program, which solves a deterministic and finite-time-horizon variant of the model. It is based on the ILP proposed by Hulshof et al. in [9]. The authors, however, never compared the ILP with the approximate dynamic programming approach proposed for the same problem in [10]; this comparison is made in this thesis. The (I)LP we have formulated should be used with a rolling time horizon: in every time period, the (I)LP is solved for an extended time period, such as one year, and only the first action is implemented. Secondly, we proposed four decision rules which provide feasible, but not necessarily optimal, solutions to the timeslot allocation problem. The rules do not take into account possible future consequences of actions. Finally, we also implemented the method currently used by SMK, which is to use static (fixed) rosters, in order to compare whether the methods proposed in this thesis can improve on the current situation.

Some experimentation was necessary to investigate effects of hyperparameters on the algorithms, choose basis functions for LSPI, and evaluate the performance of the various decision rules. For this purpose, a small and large test problem were constructed. Exact value iteration could be used to find an optimal policy for the small test problem. We could then compare which basis functions resulted in the closest approximation by the linear parametrisation used in LSPI to the true value function. Basis functions 1 (the number of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$ who have been waiting for w time periods) and 2 (the number of patients in queue $j \in \mathcal{J}$ with urgency level $u \in \mathcal{U}_j$ who are early, on time and late for treatment) resulted in the smallest difference between the true and approximate value function. Results showed that the approximate policy generated by LSPI resulted in very similar performance to the optimal policy generated by EVI, although not strictly better than the rolling-horizon LP.

The large test problem provided more information on the behaviour of LSPI and the rolling-horizon LP regarding hyperparameters. Experiments confirmed that basis function 1 should be used for LSPI. We saw that increasing the size of the dataset, although it increased the running time of the algorithm, improved convergence and performance of LSPI up to a point, after which performance plateaued. Many other alterations were attempted to improve performance and convergence of the algorithm, but none were successful. Both LSPI and the rolling-horizon LP use a discount factor, γ, in the contribution function. In the case of LSPI, increasing γ resulted in better performance, as future effects of actions were taken into account more in the value function. However, for $\gamma \geq 0.8$, the algorithm no longer converged. On the large test problem, we found that as long as we chose $\gamma \in [0.3, 0.95]$, performance did not change much, and was higher than levels outside this range. On the real-world data, we saw that increasing γ also improved performance of the rolling-horizon LP, although this increase plateaued after $\gamma = 0.6$. Finally, the large test problem showed us that the Highest Contribution decision rule, which treats patients in order of highest cost + reward, effectively maximizing the immediate contribution gained in the current time period, performed best out of all decision rules.

The knowledge gained from these test problems could then be applied to the real-world data provided by SMK. Here, we also applied the planning-ahead portion of the problem. The method developed to predict the state P time periods ahead was tested and showed a roughly linear increase in the difference between predicted and realized state. When planning 12 weeks ahead, approximately 20% of the patients are in different queues than expected. The increased difference between true and predicted state has a negative impact on the performance of the solution methods. However, for the rolling-horizon LP and Highest Contribution rule, this performance drop is not noticeable until planning more than 6 weeks ahead. The rolling-horizon LP resulted in the highest contribution gained over one year, with the decision rule coming in second place. Although LSPI performed best on the large test problem, it was worse than the other two methods on real-world data, and even performed worse than the static roster method when planning more than 10 weeks ahead. We suspect that the increased size and complexity of the problem no longer permits a good solution in the form of a linear parametrisation.

We exploited the performance increase caused by planning less far ahead by proposing a hybrid method, where α% of OD appointments are fixed in advance by a static roster, and the remaining ap-

pointments are planned τ time periods ahead using the rolling-horizon LP. Experiments showed that $\alpha = 60\%$ and $\tau = 3$ (6 weeks) resulted in a high average contribution while ensuring that as many appointments as possible can be booked further in advance. This hybrid method was compared to the static roster and rolling-horizon LP with a planning horizon of 12 weeks to determine the actual impact on reduction of access times and resource capacity waste. The configuration of cost and reward parameters used resulted in a 23% increase in the number of FC appointments within the urgency level deadline, and a 5% increase for DC appointments, compared to the static roster method. For other appointment types, small increases in the number of on-time appointments were also seen. Adjusting the cost and reward parameters changes which patients are prioritised by the LP and will therefore lead to different results. Finally, we should note that better results can be expected from the hybrid method and rolling-horizon LP when the integer constraint is applied to actions.

11.2. Recommendations

In Section 11.2.1, we provide recommendations specifically for the Sint Maartenskliniek based on our findings in Chapter 10. Recommendations for future research on improvements to the solution methods are discussed in Section 11.2.2.

11.2.1. Practical recommendations

We recommend using the hybrid method with either the rolling-horizon ILP or the Highest Contribution decision rule, as these performed much better than LSPI on the real-world problem, are easier to implement and are very fast. Note that we use the term rolling-horizon *ILP* here, as, for practical use, the integer constraint on actions should be applied in the program to ensure optimality and less wasted resource capacity. The parameters $\alpha = 60\%$ and $\tau = 3$ can be used with the hybrid method, but should be confirmed using data of other surgeons. In practice, the capacity planning department should use a roster in which 60% of appointments are determined by the static roster method when planning OD sessions twelve weeks in advance. Patients can already be booked into these timeslots. The remaining 40% of appointments can then be allocated using one of the two dynamic methods six weeks in advance. The required input will be the current state and the rosters in the five upcoming weeks to determine the predicted state. Resource capacities for that time period are variable input to determine a feasible action. OD and OR time capacities for later time periods, rather than average capacities, will also improve the prediction of the influence of the selected action on future weeks.

The choice between the rolling-horizon ILP and decision rule depends on the software available to the hospital. The state prediction algorithm, decision rule and simulation can be programmed in a standard programming language. For this thesis, all code was written in Python, but another programming language can be used for implementation. However, to solve the ILP, the hospital must have access to a solver such as Gurobi, which was used in this thesis, or AIMMS. If this is not the case, we recommend using the decision rule, as the performance decrease is not significant, and solvers cost money. Another option is to relax the integer constraint on actions and use an open-source solver, as these are capable of solving linear programs.

Before the hybrid method can be implemented, data must be collected for all surgeons to determine transition probabilities and the number of new patients entering the system per time period. We extracted this information from logistic path data of the hospital. During the data analysis phase, it became apparent that a lot of data was missing from the logistic path data; in particular, many OR appointments seemed to be missing. To accurately determine transition probabilities to use for real-life planning, this data must be updated. Furthermore, these parameters should be recalculated every six months in order to reflect any structural, long-term changes that could arise in patient treatment.

For simplicity, the data used in this thesis came from a surgeon who worked almost exclusively within one unit and did not treat many special patients. Most surgeons at SMK work for multiple units, and transition probabilities differ between units. In that case, new queues should be added to the model for each unit. Furthermore, if a significant portion of a surgeon's patients are special patients, a special queue should be added for each appointment type as transition probabilities and urgency levels for

these patients are different compared to normal patients as well. These additional queues will increase the size of the model and therefore increase solving time, but since both the ILP and decision rule are very fast, it is unlikely that this will cause difficulties for the capacity planning department.

The costs and rewards used by the dynamic methods must be set by the hospital. Since the influence of these parameters on access times for different appointment types is not immediately clear from the values alone, we suggest performing a similar simulation as in Section 10.4, and adjusting the costs and rewards until a desirable result is achieved. Another method is to use the iterative weight-updating procedure proposed by Hulshof et al. in [9]. There, weights (costs/rewards) in the objective function are initialised randomly, the corresponding MILP is solved, and performance metrics are measured from the solution. This analysis is then used to update the weights, and the process is repeated. Alternatively, costs and rewards can be assigned according to the actual financial costs for wasting resource capacity and increasing waiting times.

11.2.2. Future research

Although many adaptations to LSPI were attempted in this thesis to improve performance and convergence, many more can be identified. These are presented in Section 11.2.2.1. Our other recommendations concern further research on the rolling-horizon ILP, and are discussed in Section 11.2.2.2.

11.2.2.1. Recommendations for LSPI

In Section 9.2.1, we learned that replacing random states in the dataset with states visited in the simulation negatively impacted performance. Another alternative, to replace random states with "bad" states (for example, where many patients have high waiting times), was not attempted. It would be interesting to find out whether this teaches the algorithm to avoid such situations, possibly improving performance. We also saw that, as problem size increased, performance of LSPI relative to other methods declined. A reason for this could be that the problem complexity cannot be handled by a linear parametrisation of the value function. This provides some motivation to look into a non-linear parametrisation, or non-linear basis functions. However, in that case we would not be able to use the expected state or the policy-LP, and a different method would have to be used to find a feasible action maximizing the objective function. Any other method is likely to be slower than the policy-LP, slowing down the algorithm, which was already very slow to converge.

In Section 10.2.3, we also saw that LSPI suffered much more from using the predicted state than the other dynamic methods. This was unexpected, as LSPI and the (I)LP are designed to find a feasible action maximizing the same objective function, the same predicted state is used by both, and the difference in performance without planning ahead was much smaller. One way to improve on this could be to incorporate the state prediction step within the LSPI algorithm. Rather than having the basis function represent features of the predicted state, the basis function could represent features of the current state and the actions to be performed in the meantime. Based on this information, and the parameter vector learned during training, the policy chooses an action. Again, a downside is that this will significantly increase running time, not only due to the large increase in the number of features, but also as we need to simulate P actions and states to determine the resulting contribution for each state in the dataset.

Another way in which LSPI could be improved, is by more accurately determining the distribution of patient waiting times. LSPI is dependent on a dataset of randomly generated states, for which it is necessary to draw waiting times of patients from a probability distribution. In this thesis, these distributions were very roughly estimated, resulting in inaccurately generated states. It is likely that the algorithm would benefit from a more realistic dataset.

Another recommendation is to look into a different approximate dynamic programming algorithm which can be applied to finite-time-horizon problems, rather than infinite-time-horizon as was used throughout this thesis. This is motivated by the fact that best results for LSPI were achieved when γ was set relatively low. With $\gamma = 0.6$, the contribution in time period 10, for example, has a weight of only $0.6^{10} = 0.006$ in the contribution function. Knowing this, we could leave later time periods out of the contribution function. Algorithms for finite time horizon problems tend to be somewhat simpler

and more reliable, so this could achieve better results than LSPI.

However, even if we were able to improve performance of LSPI, there are still three major drawbacks to the algorithm: complexity of implementation, running time, and explainability. First, the algorithm requires much more data collection in order to program a simulation, and is tricky to implement. Convergence is very sensitive to settings of hyperparameters and problem size. Secondly, on realistic problem sizes, the algorithm takes hours to converge, making experimentation with hyperparameters practically impossible. For practical use, the algorithm would have to be run every few months, to ensure model parameters are up-to-date. The high running time also implies that we cannot hope to expand the model to optimize for multiple surgeons simultaneously. Maybe the largest drawback for practical implementation is that the algorithm is a black box: we cannot reasonably explain why one action is chosen over the other. In healthcare applications, this means that we cannot explain why we accept one patient, but reject the other. This is definitely not the case for the decision rule (treat the most urgent patient), and for the LP we can argue that the actions chosen are the optimal solution to an explainable problem.

11.2.2.2. Recommendations for the rolling-horizon LP

One of the simplifying assumptions placed on the model was that a surgeon is not affected by activities of other surgeons. This allowed us to create a single-surgeon model. In real life, this is not the case. For example, at SMK, new patients are not assigned to a single surgeon, but to a treatment unit. Therefore, if one surgeon has a long RC waiting list, another surgeon can compensate by accepting more new patients. The LP formulated in this thesis can be expanded to model a unit, or even the entire orthopaedic department. This would make the program much larger, and integer constraints on variables would be necessary in order to allocate whole patients to surgeons, significantly increasing solving time. However, this could be easier for the capacity planning department as the model would output rosters for all surgeons simultaneously, rather than having to repeatedly solve the LP for every surgeon separately. Furthermore, being able to take interaction effects into account could significantly improve performance.

The fact that the Highest Contribution decision rule overtook the rolling-horizon LP in terms of average contribution when there was a high number of patients initially in the system, suggested that the rolling-horizon LP could be improved by decreasing γ when the number of patients increases. More research could be done on developing an adaptive-γ scheme for this problem. We should warn that the need to decrease the discount factor will likely not only rely on the number of patients, but also on the available resource capacity, which was constant within this thesis but variable in general real-world situations. This is because the judgement of whether there are “too many” patients is relative to the number of available timeslots in which to schedule those patients. Nevertheless, this is a relatively simple adjustment to the linear program which does not greatly affect running time and is guaranteed to improve performance.

Bibliography

[1] Rijksoverheid. Het coronavirus en de zorg in ziekenhuizen, 2021. URL https://www.rijksoverheid.nl/onderwerpen/coronavirus-covid-19/gezondheid-en-zorg/ziekenhuizen.

[2] Yogita Limaye, Shalu Yadav, and Aakriti Thapar. Coronavirus: Overwhelmed India hospitals turn Covid patients away, 2020. URL https://www.bbc.com/news/av/world-asia-india-53014213.

[3] Nederlandse Zorgautoriteit. Zorgplicht zorgkantoren – beeld 2019 en uitdagingen (deel 3 bij Onderzoeksrapporten toezicht op langdurige zorg). Technical report, 2020. URL http://puc.overheid.nl/doc/PUC_624861_22.

[4] Orthopedie. URL https://www.maartenskliniek.nl/orthopedie.

[5] Stichting Sint Maartenskliniek. Jaarverslag 2019. Technical report, 2020.

[6] Rhythm. URL www.rhythm.nl.

[7] Marelise Hattingh. Decision support for planning and scheduling orthopaedic surgeons. Technical report, University of Twente, 2019.

[8] Eline R. Tsai. Optimal time allocation of an orthopedic surgeon. Technical report, University of Twente, 2017.

[9] Peter J.H. Hulshof, Richard J. Boucherie, Erwin W. Hans, and Johann L. Hurink. Tactical resource allocation and elective patient admission planning in care processes. *Health Care Management Science*, 16(2):152–166, 2013. ISSN 13869620. doi: 10.1007/s10729-012-9219-6.

[10] Peter J.H. Hulshof, Martijn R.K. Mes, Richard J. Boucherie, and Erwin W. Hans. Patient admission planning using Approximate Dynamic Programming. *Flexible Services and Manufacturing Journal*, 28(1-2):30–61, 2016. doi: 10.1007/s10696-015-9219-1.

[11] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4(6):1107–1149, 2003. doi: 10.1162/1532443041827907.

[12] Warren B. Powell. *Reinforcement learning and stochastic optimization problem: A unified framework for sequential decisions*. John Wiley & Sons, Hoboken, New Jersey, 2019.

[13] Nederlandse Zorgautoriteit. Beleidsregel TH / BR-018 Toezichtkader zorgplicht zorgverzekeraars Zvw, 2014. URL https://puc.overheid.nl/nza/doc/PUC_21398_22/1/.

[14] Erwin W. Hans, Mark Van Houdenhoven, and Peter J.H. Hulshof. A Framework for Healthcare Planning and Control. In *Handbook of Healthcare System Scheduling*, pages 303—-320. Springer US, Boston, MA, 2012. doi: 10.1007/978-1-4614-1734-7_12.

[15] Nikky Kortbeek. Integrale capaciteitsplanning van de orthopedische keten in de Sint Maartenskliniek, 2019.

[16] Olivier Sigaud and Olivier Buffet. *Markov decision processes in artificial intelligence*. John Wiley & Sons, 2013.

[17] Marco Wiering and Martijn Van Otterlo. *Reinforcement Learning*, volume 12. Springer, Berlin, 2012. doi: 10.1007/978-3-642-27645-3.

[18] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[19] Lucian Busoniu, Bart De Schutter, and Robert Babuska. Approximate dynamic programming and reinforcement learning. *Studies in Computational Intelligence*, 281:3–44, 2010.

[20] Paul J. Schweitzer and Abraham Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110(2):568–582, 1985. doi: 10.1016/0022-247X(85)90317-8.

[21] Leemon Baird. Residual Algorithms: Reinforcement Learning with Function Approximation. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 30–37, Tahoe City, California, jan 1995. Elsevier. doi: 10.1016/b978-1-55860-377-6.50013-x.

[22] Steven J. Bradtke and Andrew G. Barto. Linear Least-Squares Algorithms for Temporal Difference Learning. *Machine Learning*, 22:33–57, 1996.

[23] Peter C. Young. *Recursive Estimation and Time-Series Analysis*. Springer-Verlag, 1984.

[24] Alessandro Lazaric, Mohammad Ghavamzadeh, and Remi Munos. Analysis of a Classification-based Policy Iteration Algorithm. In *27th International Conference on Machine Learning*, pages 607–614, Haifa, Israel, 2010.

[25] Lei Zuo, Qi Guo, Xin Xu, and Hao Fu. A hierarchical path planning approach based on A* and least-squares policy iteration for mobile robots. *Neurocomputing*, 170:257–266, 2015. doi: 10.1016/j.neucom.2014.09.092.

[26] Lihong Li, Jason D. Williams, and Suhrid Balakrishnan. Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection. In *Proceedings of the 10th Annual Conference of the International Speech Communication Association (INTERSPEECH 2009)*, pages 2475–2478, Brighton, UK, 2009.

[27] Jian Wang, Xin Xu, Daxue Liu, Zhenping Sun, and Qingyang Chen. Self-Learning Cruise Control Using Kernel-Based Least Squares Policy Iteration. *IEEE Transactions on Control Systems Technology*, 22(3):1078–1087, 2014. doi: 10.1109/TCST.2013.2271276.

[28] Hans Jörg Schütz and Rainer Kolisch. Approximate dynamic programming for capacity allocation in the service industry. *European Journal of Operational Research*, 218(1):239–250, 2012. doi: 10.1016/j.ejor.2011.09.007.

[29] George B. Dantzig and Mukund N. Thapa. The Linear Programming Problem. In *Linear Programming 1: Introduction*, chapter 1, pages 1–25. Springer Science & Business Media, 2006.

[30] Laurence A. Wolsey. Formulations. In *Integer Programming*, chapter 1, pages 1–19. John Wiley & Sons, Inc., 1998.

[31] Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. 51(3):296–305, 2001. doi: 10.1145/380752.380813.

[32] Gurobi Optimization LLC. Gurobi Optimizer Reference Manual, 2021. URL `https://www.gurobi.com`.

[33] Oskar Roemeling, Kees Ahaus, Folkert Van Zanten, Martin Land, and Patrick Wennekes. How improving access times had unforeseen consequences: A case study in a Dutch hospital. *BMJ Open*, 2019. doi: 10.1136/bmjopen-2019-031244.

[34] Konstantinos Koutroumbas and Sergios Theodoridis. *Pattern Recognition*. Academic Press, 2008. ISBN 9780080949123.

[35] Dimitri P. Bertsekas. Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications*, 9(3):310–335, 2011. doi: 10.1007/s11768-011-1005-3.

[36] Lihong Li, Michael L. Littman, and Christopher R. Mansley. Online exploration in least-squares policy iteration. In Decker, Sichman, Sierra, and Castelfranchi, editors, *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 733–740, Budapest, Hungary, 2009.