Networks of Sensors

Operation and Control

Martijn Onderwater
Networks of Sensors – Operation and Control
ISBN: 978-94-028-0004-3

The research in this thesis has been carried out as part of the project Realisation of Reliable and Secure Residential Sensor Platforms (RRR) of the Dutch program *IOP Generieke Communicatie*, number IGC1020, supported by the *Subsidieregeling Sterktes in Innovatie*.

Permission for the cover design was granted by the *Standaard Uitgeverij*, publisher of the comic "Suske en Wiske".

Printed by Ipskamp Drukkers, The Netherlands.

VRIJE UNIVERSITEIT

Networks of Sensors

Operation and Control

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. V. Subramaniam,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Exacte Wetenschappen
op maandag 8 februari 2016 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Martijn Onderwater

geboren te Leiderdorp

promotor: prof.dr. R.D. van der Mei
copromotor: dr. S. Bhulai

Dankwoord (Acknowledgements)

Dit proefschrift is het resultaat van vijf jaar noeste arbeid aan het Centrum voor Wiskunde & Informatica (CWI) en de Vrije Universiteit (VU) te Amsterdam. Zelfs al staat mijn naam als enige op de kaft, het onderzoek zou nooit tot stand zijn gekomen zonder de hulp en steun van een grote groep mensen.

Allereerst gaat mijn dank uit naar Rob van der Mei. Rob, je gaf me de tijd en de ruimte om mijn onderzoek vorm te geven, en daar heb ik dankbaar gebruik van gemaakt. Met je eindeloos optimistische insteek heb je me gaande gehouden tijdens de onvermijdelijke wetenschappelijke dipjes, en met de juiste vragen hielp je me het onderzoek kwalitatief een stap verder te brengen. Daarnaast zullen jouw communicatieve vaardigheden nog lang een inspiratiebron voor mij zijn.

Een groot deel van mijn onderzoek is gedaan in samenwerking met Sandjai Bhulai. Sandjai, ons teamwork is me uitstekend bevallen en ik heb enorm veel geleerd over Markov beslissings-modellen en het schriftelijk overbrengen van mijn onderzoek. Je positieve en creatieve instelling heeft onze gezamenlijke papers naar een hoger vlak getild.

Mijn collega's bij het CWI en de VU zijn eveneens een essentieel onderdeel geweest van de afgelopen vijf jaar. De sociale cohesie tussen collega's is sterk, en ik vermoed dat velen van hen jaren later nog vrienden van me zijn. Dankzij jullie heb ik ook het hardlopen ontdekt, en daar heb ik iedere week nog profijt van.

Het onderzoek is in financiële en praktische zin ondersteund door de Rijksdienst voor Ondernemend Nederland en de partners in het RRR project: Munisense, Thales Nederland, Tyco Integrated Fire & Security, de Universiteit Twente, en het CWI. In het bijzonder wil ik Gerard Hoekstra, Maurits de Graaf, Mihaela Mitici, Joost Pastoor, George Boersma, Bart-Jan Coolen, en Cor Aalberts

bedanken voor de prettige samenwerking en de interessante inhoudelijke discussies. Verder wil ik Ger Koole bedanken voor de gastvrijheidsverlening bij de VU.

Tot slot wil ik mijn familie en vrienden bedanken voor de geboden steun door de jaren heen. Wellicht zeg ik dit niet vaak genoeg hardop, maar jullie vertrouwen en advies stellen me in staat om 'op koers' te blijven en om dit soort langetermijntrajecten met succes én plezier te volbrengen. Verder realiseer ik me dankzij jullie steeds meer dat het maken van een cryptogram, doelloos dwalen bij Office Center, The Good Wife kijken, praten over sport(club) en voeding, lunchen met een tante, herinneringen ophalen over de clown Popov, en een documentaire over vampieren kijken óók de moeite waard zijn om tijd voor te maken.

Contents

1

Introduction

Over the past few years, the use of sensors has been growing at an unprecedented rate, and sensors play an increasingly large role in our daily lives. Smartphones, for example, contain a wide variety of sensors. The documentation of Android's API [10] reveals support for sensors measuring acceleration, temperature, relative humidity, air pressure, degrees of rotation, the geomagnetic field, gravity, illumination, and proximity. Apps use these sensors to, e.g., detect shaking (shuffle songs), determine rotation of the screen (prevent recording vertical video), measure a user's level of activity (in a fitness app), and recognize proximity to an ear (switch off the touchscreen during a call).

Another interesting application of sensors is in modern washing machines, in which sensors monitor several aspects of the washing cycle, including water hardness, the amount of dirt and grease, the type of detergent used (liquid/powder), and the weight of the laundry. These observations are then used to control the process, allowing the machine to, e.g., add clean water, change the direction and speed of the spin, and increase the water's temperature. By controlling these parameters, the washing machine can optimize the length of washing cycle, minimize energy usage, and prevent damage to components.

The increasing popularity and relevance of sensors is also evident from Gartner's annual 'Emerging Technologies Hype Cycle' [12]. This graph, shown in Figure 1.1, reflects the level of maturity of a technology during its lifetime (horizontal axis), compared to expectations attached to it by society (vertical axis). The left side of the graph contains innovative, promising technologies that are still fairly unknown and have low expectations. The next stage is character-

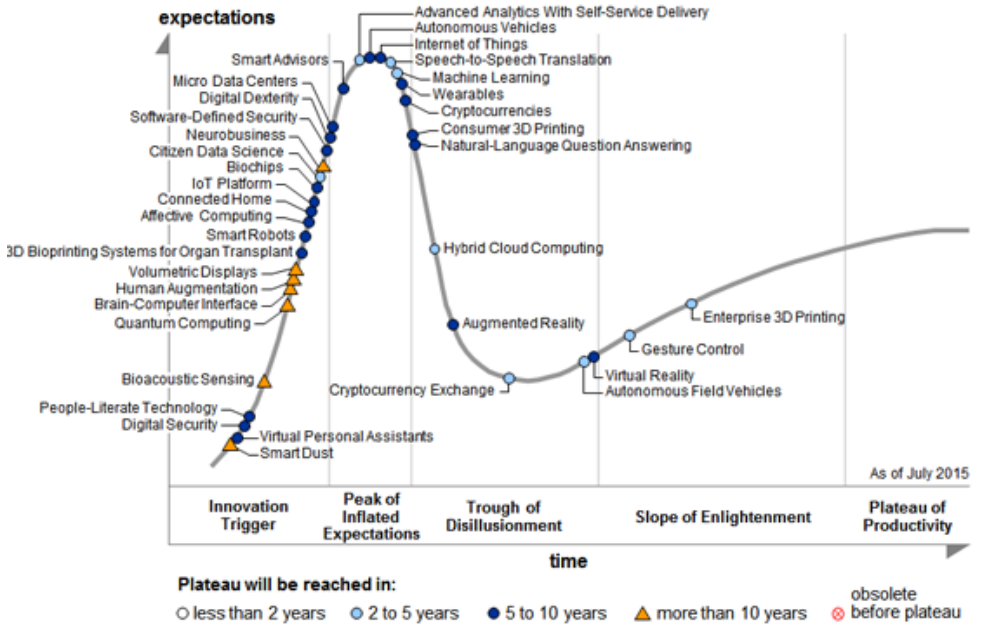


FIGURE 1.1: Gartner's 2015 Hype Cycle of Emerging Technologies [12].

ized by a massive peak of inflated expectations, while the technology is still relatively immature. Typically, this is where a technology reaches the status of 'hype'. Inevitably, this leads to some disillusionment when people realize that expectations about the technology are overrated. Eventually, increased understanding, knowledge, and experience drive the technology towards maturity at a more moderate pace.

Several of the technologies presented in the hype cycle are closely related to sensors. At the top of the hype cycle is 'The Internet of Things' (IoT), a concept referring to a network of physical objects that allow interaction between the physical world and computer-based systems. 'Things' can be, e.g., coffee machines, curtains, smart energy meters, security systems, or watches. Sensors play a central role in IoT, because they are small, energy efficient, have (wireless) connectivity, and still carry sufficient computational resources for basic applications. This makes them ideal to connect 'Things' to 'The Internet'. Moreover, recent technological developments have resulted in cheap and reliable sensors, making them cost-efficient for use in applications, and a driving factor for IoT. Furthermore, note that the term 'IoT Platform' is on the hype cycle as well. This platform facilitates collecting, analyzing, and integration of data from 'Things'. In Chapter 2 we discuss this technology in more detail in a sensor-specific context.

Slightly further along in maturity is 'Wearables', of which the smart watch is probably the most well-known example. The typical smart watch contains several sensors, such as an accelerometer, a thermometer, an altimeter, a barometer, a compass, GPS, and a heart rate sensor. Close integration with smart phones allows wearers to, e.g., play music, get social media notifications, and track physical activity. The Wearables technology is about to enter the third phase on the hype cycle (the 'through of disillusionment'), so it will be interesting to see how it develops in the near future. Other technologies on the hype cycle related to sensor technology are 'Smart Dust' (millimeter-sized devices with embedded sensors), 'Bioaccoustic sensing' (enables the use of, e.g., your arm as a touchpad), and 'Connected Home' (home automation).

The data-driven technologies '[Citizen] Data Science', 'Advanced Analytics [With Self-Service Delivery]'[1], and 'Machine Learning' on the hype cycle are relevant for sensor data as well. Sensors can produce massive amounts of data, because of the potential high frequency of measurements, and the large number of relevant phenomena to monitor. For instance, a typical Boeing aircraft contains 8,000-10,000 sensors that are measuring and reporting every second [39]. Extrapolate this to 5,000 aircraft demonstrates that Boeing receives over fourteen terabyte of sensor data each day. Dealing with such large amounts of data and transforming it to actionable insights is an immense challenge.

1.1 Networks of sensors

The sensors in a smartphone are capable of measuring a large variety of phenomena, both concerning the device itself (e.g., detecting the rotation of the screen), as well as about the immediate environment (e.g., the ambient temperature). Measuring the temperature in, e.g., an office building with a smartphone is, however, unpractical because it requires the phone's owner to continually walk around the building recording the current temperature. Fortunately, with modern day technology it is possible to make small 'mini-computers' to which sensors can be attached, yielding devices smaller than a credit card capable of monitoring of and interaction with the environment. In this thesis we refer to these devices as *sensor nodes*. Typically, besides sensors and basic processing capabilities, a node is also equipped with a small radio that allows it to

[1] Citizen Data Science refers to the fact that improved data science tools allow the average Joe to apply data science, eliminating the need for highly trained data scientists. Advanced Analytics With Self-Service Delivery means that modern, simple to use Business Intelligence tools allow people to analyze data without extensive need of IT support.

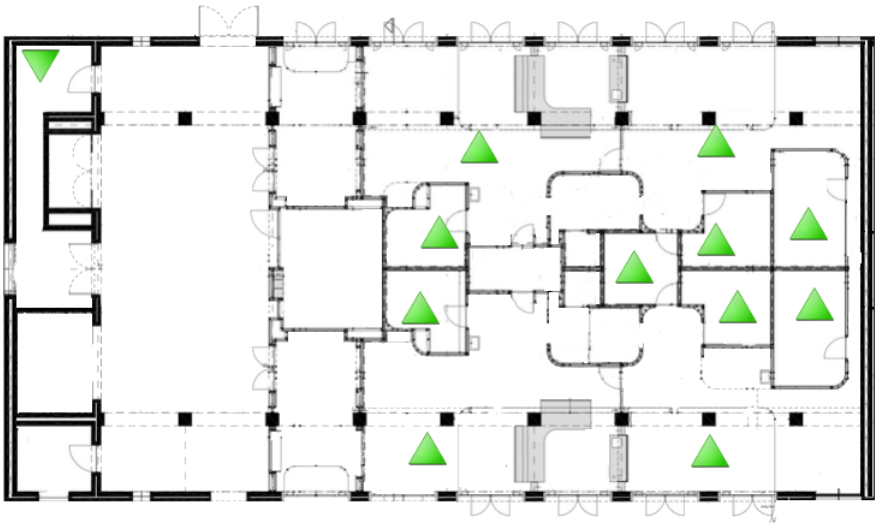


FIGURE 1.2: Floor-plan of a kindergarten, with sensor nodes (triangles) measuring the indoor climate and reporting their measurements to a central node (inverted triangle).

communicate wirelessly. Hence, sensor nodes can report their measurements to interested applications, and thus facilitate remote monitoring and control. Obviously, this is of tremendous practical value when the sensor nodes are spread over a large geographical area. However, the range of the radio of a sensor node is usually small in order to save energy, so communicating across large distances is not possible. As a solution, protocols have been developed that allow the sensor nodes to jointly form a network, so that data can be transmitted over long distances in several smaller steps. These networks are called *wireless sensor networks* (WSNs), and are the topic of the first part of this thesis.

An example deployment of a WSN is in Figure 1.2, showing the floor-plan of a kindergarten. This kindergarten is located in a city in The Netherlands, where directives are in place with respect to the indoor climate. For a kindergarten, a CO_2 level larger than 1,000 parts per million (ppm) is used as an indication of insufficient ventilation. High levels of CO_2 are associated with fatigue, headaches, and reduced concentration [11, 43]. The sensor nodes, marked by triangles, measure CO_2, temperature, humidity, and illumination. The measurements are transmitted wirelessly to a so-called *sink node* (inverted triangle), where the data is collected for further processing. After processing, the administrators of the kindergarten can monitor the current CO_2 level using graphs similar to the one displayed in Figure 1.3. It shows the CO_2 levels

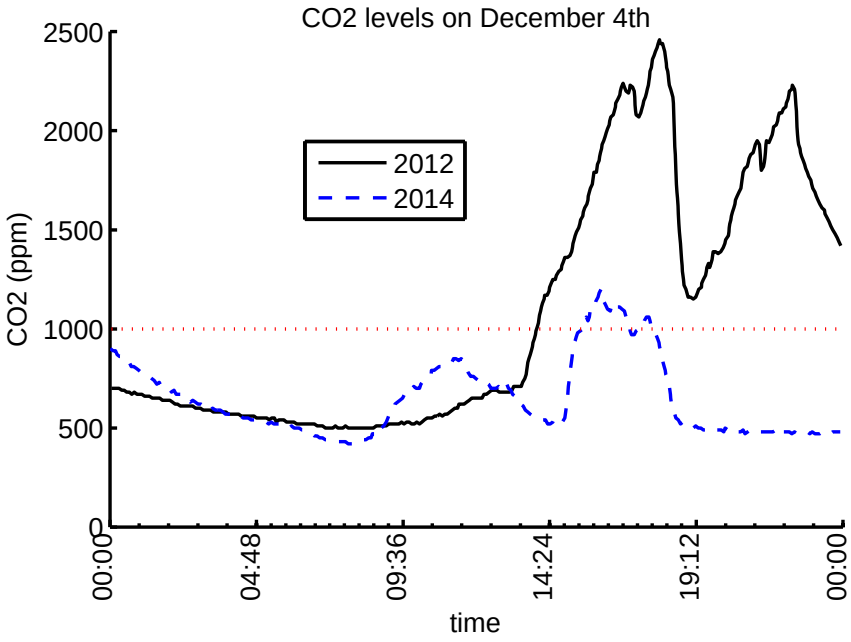


Figure 1.3: CO_2 levels in the kindergarten exceeding the recommended level of 1,000 particles per million on December 4th 2012, and a much healthier development on the same day in 2014.

on December 4th 2012 (solid line), on December 4th 2014 (dashed), and the maximum threshold value of 1,000 ppm (dotted). The 2012 line demonstrates that CO_2 levels can exceed the 1,000 ppm threshold, and the 2014 line shows much healthier levels.

1.2 Challenges

Driven by emerging technologies, applications relying on sensor data will become more prevalent in the near future. However, working with sensors involves some distinct challenges that have to be dealt with. For instance, sensor measurements are often 'noisy' and can include a significant level of uncertainty. In the kindergarten in Figure 1.2 several sensor nodes were installed upside down (as compared to the other nodes), because this was more convenient for the power cable. Afterwards we noticed that the temperature measurements at

these nodes differed by approximately 1.5° Celsius from the other nodes. The manufacturer of the nodes then confirmed that heat from the sensor node can warm up the temperature sensor if a node is installed upside down. This illustrates how a seemingly small mistake can affect sensor measurements, and how it can contribute to the uncertainty associated with these measurements. Having a network of sensor nodes raises additional challenges, especially when the network is wireless (which is common in practice). The transmission of a packet from one node to another node can easily fail because of, e.g., closed doors, people walking by, or interference on the wireless channel. In this section we discuss the most important challenges in the context of sensors, sensor nodes, and sensor networks. The challenges below are adapted from [18, 110, 132], and appear in no particular order.

- *Limited resources.* Sensor nodes typically have only few resources available, i.e., little storage capacity, a slow CPU, and a battery with limited power. Creating applications on such a device often results in a careful balancing act between satisfying application requirements and managing resources. For instance, measuring with a sensor and transmitting the resulting measurement are two of the most power-consuming operations a node can do. To maximize the life-time of the battery, the frequency of these operations should be minimized, which can be problematic for applications relying on frequent measurements.

- *Dynamic topology.* The topology of a sensor network can be highly dynamic, because sensor nodes can be added, removed, moved, run out of power, or break down. In certain applications, the sensor node might even be mobile instead of stationary. Additionally, interference on the wireless channel might cause a sensor node to be temporarily unavailable.

- *Data redundancy.* The data produced by sensor networks usually contains a large amount of redundancy. For instance, the rooms in the kindergarten in Figure 1.2 are all in the same building and on the same floor, and the temperature measurements by the sensor nodes are expected to be similar. Redundancy can also occur between different types of measurements: in a closed room full of people, both temperature and CO_2 levels are expected to increase, and thus the two types of measurements are correlated. Applications relying on large amounts of sensor data might need to reduce redundancy to remain computationally feasible. At the same time, redundancy in data can also be useful. When a certain level of redundancy is expected but, upon arrival of the measurements, is not observed, then this absence of redundancy might suggest that something happened or that there is a problem.

- *Reliability.* Measurements by sensors can contain random noise, so that it is unclear how accurate a measurement is. This is particularly challenging when monitoring an environment for abnormal events, since an inaccurate measurement and an abnormal measurement can be difficult to distinguish. Also, the wireless channel and the network are susceptible to reliability issues. The channel might be slower than expected, completely unavailable in part of the network, and the topology can change. These issues cause delay or even complete disappearance of measurements, making it challenging for the network to operate reliably.

- *Scalability.* Sensors are relatively cheap and can be used cost-effectively in large quantities. Consequently, sensor networks can potentially be quite large, and the network infrastructure should be scalable to such large sizes.

- *Heterogeneous protocols and data formats.* Sensor nodes can use a wide variety of protocols, access mechanisms, and data formats. This heterogeneity hampers development of applications, since acquiring sensor data requires a significant amount of sensor-specific work.

1.3 Motivation and structure

Sensors and sensor networks are here to stay. Even though IoT is now at the peak of inflated expectations in the hype cycle (see Figure 1.1), sensor-related technology is clearly useful in many applications. The nearing period of improved maturity of IoT suggested by the hype cycle will make the challenges from the previous section increasingly relevant. This raises the need for a deeper understanding of the challenges, for practical methods to deal with them, and for innovative solutions. This is the main motivation for the research in this thesis.

In the following chapters we consider several topics related to the challenges outlined in the previous section. The thesis consists of seven chapters (including this introduction), and is divided in two parts. The first part is about sensor technology, and discusses three different topics in that context. The relation between the three topics is illustrated in Figure 1.4, and explained in more detail in the paragraphs below. In the second part we deal with Markov Decision Processes, a popular framework for taking decisions under uncertainty. The techniques in that part are described in general terms, because the techniques can also be applied in contexts other than sensor networks. The various chap-

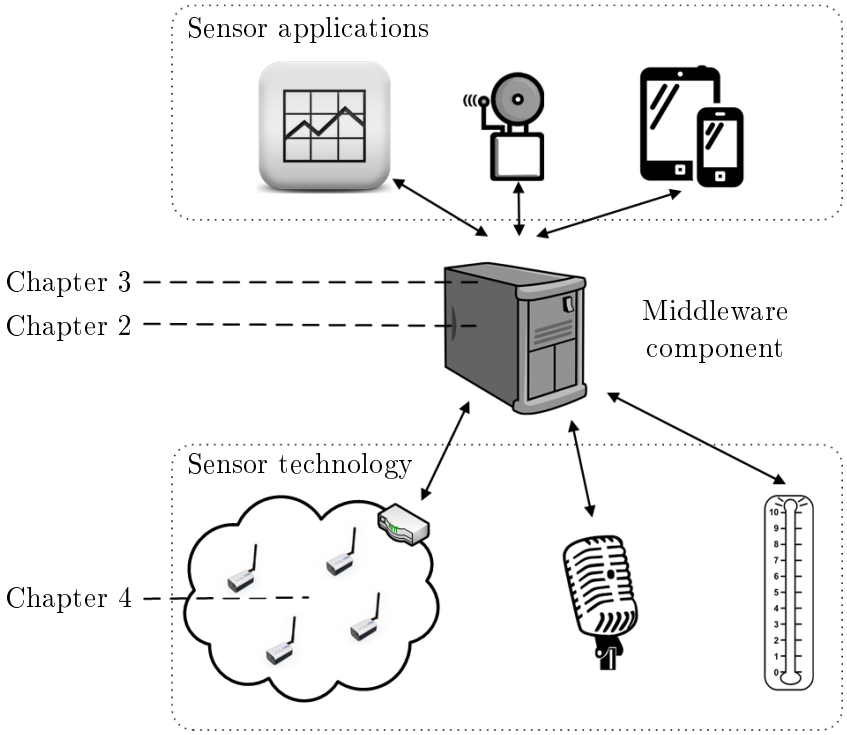


FIGURE 1.4: A middleware component forms a bridge between sensor technology and sensor applications. The figure also illustrates how the three sensor-related chapters in Part I relate: Chapter 2 is about middleware components, Chapter 3 discusses dimensionality reduction techniques (a potential service offered by a middleware component for applications), and Chapter 4 deals with throughput of a sensor network (the left-most example of a sensor technology).

ters in this thesis can be read independently of each other and in arbitrary order, with the exception of Chapter 7, which extends Chapter 6.

We start Part I with a discussion about middleware components for sensor networks in Chapter 2. These components are closely related to the 'IoT Platform' mentioned in the hype cycle in Figure 1.1. The middleware component forms a natural bridge between sensor technology and applications using sensor data. Figure 1.4 illustrates this scenario, with a sensor network and two standalone sensors (for sound and temperature) at the bottom of the figure providing measurements to the middleware component. The top of the figure shows three applications relying on sensor data: a chart, an alarm application, and a mobile app. The middleware component is in the center, and decouples

the sensor applications from the sensor technology. In the scientific literature, a wide variety of such middleware components exists and in Chapter 2 we review these components with an often-used categorization. Then we describe that, recently, a new category of middleware components has emerged and we introduce the name 'centralized' for this category. We describe the general architectural form of a centralized middleware component, review four well-known components in the new category, and discuss their relevance for use in sensor networks.

Next, in Chapter 3 we consider *dimensionality reduction techniques*, which aim at removing redundancy from data by finding a short insightful summary. These techniques can be applied to sensor data as well, allowing applications to work with only a small part of the sensor data instead of the full range of measurements. Dimensionality reduction can, e.g., be a service provided by a centralized middleware platform (illustrated in Figure 1.4). The summary resulting from dimensionality reduction is designed to retain the most important part of the information from the original data, but inevitably some information is lost. For certain applications this loss might be unacceptable. For instance, an alarm application watching for abnormal sensor measurements ('outliers') can only use the summarized data if outliers among the sensor data are mapped to outliers among the summarized data by the reduction technique. If the reduction technique does not preserve outliers, the alarm application will miss measurements worthy of an alarm when using the summarized data. In Chapter 3 we discuss three popular dimensionality reduction techniques, and experimentally determine how well they preserve outliers. The experiments identify one of the techniques as best able to preserve outliers, and we discuss the intuitions behind this result.

In Chapter 4 we consider the *saturation throughput*, an important performance indicator of a sensor network (the left-most sensor technology in Figure 1.4). This property reflects at what speed the network is able to process measurements by sensors when a large number of these measurements is offered. We develop a model for analyzing the saturation throughput of the IEEE 802.15.4 MAC protocol, which is the de-facto standard for WSNs and ensures fair access to the channel. To this end, we introduce the concept of a *natural layer*, which reflects the time that a sensor node typically has to wait (as prescribed by the IEEE 802.15.4 MAC protocol) prior to sending a packet. The model is simple and provides insight in how the throughput depends on the protocol parameters and the number of nodes in the network. Validation experiments with simulations demonstrate that the model is highly accurate for a wide range of parameter settings of the MAC protocol, and for both large and small networks.

The first chapter in Part II, Chapter 5, deals with the control of a queueing system with aging state information. The controller of the system has to provide incoming queries with a response. The response can either be a fresh value obtained from a queueing system, or an older value that was cached by the controller. Both choices are imperfect: the first causes a delay because it takes time to generate a fresh response, and the second returns an aged value that is potentially too old for use. Hence, the controller faces a trade-off between response times and data freshness. In practice, a threshold policy is often used to take decisions, where a fresh value is generated when the age of the cached response exceeds a given threshold. Unfortunately, this policy is not always optimal, particularly when the queueing system is heavily loaded, and requesting a fresh response is expensive. In Chapter 5 we demonstrate how such a threshold policy can be improved by taking the load of the system into account. We model the system as a Markov Decision Process, which turns out to be complex. We simplify the model to circumvent these complexities, and then construct a control policy. This policy is demonstrated to have near-optimal performance and achieves lower costs than the threshold policy.

Chapter 6 introduces a novel method for discovery of relative value functions for Markov Decision Processes. This method, which we call Value Function Discovery (VFD), is based on ideas from the Evolutionary Algorithm field. VFD's key feature is that it discovers descriptions of relative value functions that are algebraic in nature. In particular, the descriptions include the model parameters of the MDP. The algebraic expression of the relative value function discovered by VFD can be used in several scenarios, e.g., conversion to a policy (with one-step policy improvement) or control of systems with time-varying parameters. In Chapter 6, we give a detailed description of VFD and illustrate its application on an example MDP. For this MDP we let VFD discover an algebraic description of a relative value function that closely resembles the relative value function corresponding to the optimal policy. The discovered relative value function is then used to obtain a policy, which we compare numerically to the optimal policy of the MDP. The resulting policy has excellent performance on a wide range of model parameters.

We continue work on VFD in Chapter 7, where we demonstrate how additional information about the structure of an MDP can be included in VFD. For this we use the same MDP as in Chapter 6, and include prior knowledge that the optimal policy is of threshold type. We let VFD learn an expression for this threshold in terms of the model parameters, and numerically inspect its performance. We demonstrate that this alternative use of VFD also yields near-optimal policies, illustrating that VFD is not restricted to learning relative value functions and can be applied more generally.

1.4 Publications

This thesis is based on the following publications:

- M. Onderwater. An overview of centralised middleware components for sensor networks. To appear in *International Journal of Ad Hoc and Ubiquitous Computing*, 2015.
- M. Onderwater. Outlier preservation by dimensionality reduction techniques. *International Journal of Data Analysis Techniques and Strategies*, 7(3):231–252, 2015.
- M. Onderwater, G. J. Hoekstra, and R. D. van der Mei. Throughput modeling of the IEEE MAC for sensor networks. *Under review*, 2015.
- M. Onderwater, S. Bhulai, and R. D. van der Mei. On the control of a queueing system with aging state information. *Stochastic Models*, 31(4): 588–617, 2015.
- M. Onderwater, S. Bhulai, and R. D. van der Mei. Value Function Discovery in Markov Decision Processes with Evolutionary Algorithms. To appear in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2015.
- M. Onderwater, S. Bhulai, and R. D. van der Mei. Learning optimal policies in Markov Decision Processes with Value Function Discovery. *Performance Evaluation Review*, 43(2):7–9, 2015.
- M. Onderwater, S. Bhulai, and R. D. van der Mei. Discovery of structured optimal policies in Markov Decision Processes. *Under review*, 2015.
- M. Mitici, M. Onderwater, M. de Graaf, J. van Ommeren, N. van Dijk, J. Goseling, and R. J. Boucherie. Optimal query assignment for wireless sensor networks. *International Journal of Electronics and Communications*, 69(8):1102 – 1112, 2015.

Part I

Sensor Networks

2

An Overview of Centralized Middleware Components for Sensor Networks

As described in the previous chapter, sensors are an emerging technology and will play a large role in our daily lives in the near future. In particular, the popularity of IoT is a driving factor for sensor-related applications. As discussed in Section 1.2, these applications will encounter several challenges that are typical for dealing with sensors, sensor nodes, and sensor networks. In traditional IT systems, a *middleware component* is often used to help applications deal with these challenges. The component forms a bridge between applications and sensor technology, and facilitates simpler application development.

The scientific literature contains a wide variety of such middleware components, based on ideas from, e.g., database technology, and on aspects of quality of service. Recently, much attention in literature is aimed at a special class of components that consider sensor networks that have no capacity to run part of the middleware component on the sensor nodes. In this chapter we introduce the term 'centralized' for these components, and illustrate their relevance using a literature review of existing middleware components. After a close look at non-centralized components, we describe the general architectural setup of a centralized component, and discuss four well-known examples of such a component. Finally, we identify directions for further research that will impact centralized components in the near future.

This chapter is based on the results presented in [3].

2.1 Introduction

Figure 2.1 illustrates the role of a middleware component in the context of sensors and of applications based on their data. On the left, sensors are used to, e.g., monitor the indoor climate of a house and detect smoke in an office building. The sensors report to the middleware component, and this component makes the data available to applications. The fire department can use this data to receive alarms related to the detection of smoke in the office building. Also, the owner of the house can inspect a graph of the current CO_2 levels, for which the data is obtained from the middleware component.

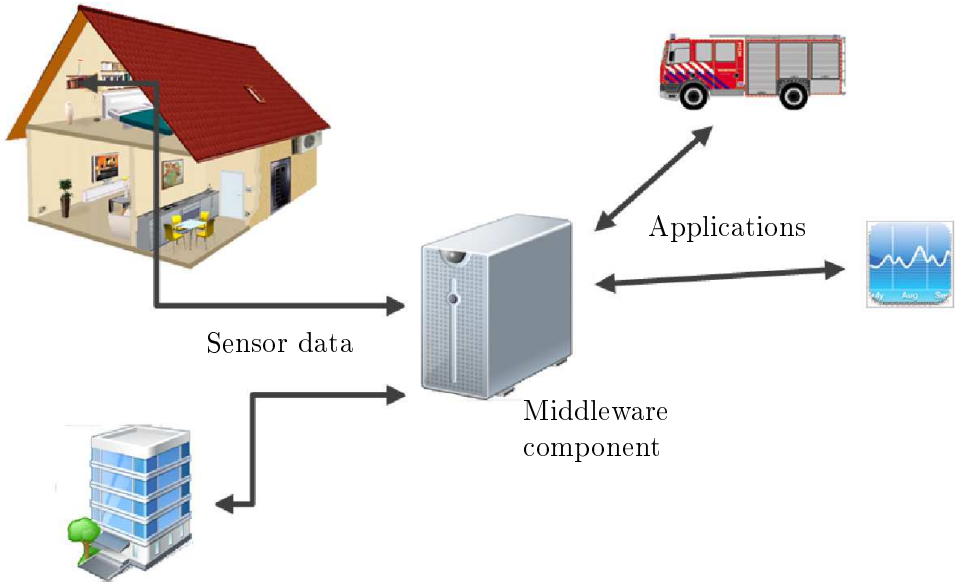


FIGURE 2.1: Illustration of how a sensor middleware component forms a natural bridge between sensors (on the left) and applications (on the right).

The middleware component forms a bridge between the sensor technology and the applications relying on sensor data. Because of this central role, the middleware component is in an ideal position to shield applications from some of the sensor-related challenges listed in Section 1.2. For instance, sensor nodes have limited resources and are often battery-powered. Excessive communication with a sensor node can quickly deplete the battery and render the node useless. A middleware component can avoid this by, e.g., providing a caching mechanism and using this to manage communication with the sensor node.

Figure 2.1 suggests that the middleware component is separated from the sensors. In this chapter we focus on components that adhere to this separation, and we introduce the term 'centralized middleware component' for them. There are, however, other approaches as well, where middleware components extend into a network of sensors and add additional intelligence to the network. These 'non-centralized' components typically require above-average computational resources, and are motivated by rapid technological developments in recent years. Despite these developments, many current sensor nodes are still simple and limited devices, and research on middleware components has slowly shifted to centralized components. Motivated by this, we review centralized middleware components for sensor networks available in the scientific literature in this chapter.

The remainder of the chapter is structured as follows. We start with a review of non-centralized middleware components in the scientific literature in Section 2.2. Then, Section 2.3 discusses the increased popularity of centralized components in recent years, and illustrates their architectural setup with a high-level outline. Following this, in Section 2.3.1, is a description of a leading web-standard for modeling sensors and, in Section 2.3.2, an overview of the most well-known centralized middleware components. Finally, we identify directions for future research in Section 2.4, and conclude the chapter in Section 2.5.

2.2 Non-centralized middleware components

Middleware components for sensor networks have received abundant attention in the scientific literature. In this section, we give an overview of the available non-centralized sensor middleware components in the literature, with an often-used categorization as a guideline. For a more in-depth review we refer to, e.g., the surveys [110, 158], or to the specialized surveys [109] (on service-oriented middleware components) and [147] (on programming sensor networks). The following categorization, adapted from [21, 60, 61, 66, 104], is often used in the literature:

- Database-inspired components,
- Virtual Machine-motivated components,
- Agent-based components,
- Application-driven components,
- Message-oriented components.

Below, we describe each category, illustrate the architectural similarities within a category, and list several middleware components contained in each category. Table 2.1 contains the middleware components per category.

Database-inspired components. This subclass of middleware components views the sensor network as a distributed database and adapts existing querying techniques to the sensor network. Figure 2.2 illustrates a simple network with three sensor nodes, and a laptop interested in collecting data from these nodes. Each node typically has a local database (DB) and a query engine for dealing with queries. Together, the storage and query processing facilities form the middleware component. As an example, suppose that the laptop issues a query to the network, requesting all measurements from nodes 2 and 3 at intervals of 1 second for the next 10 seconds. In SQL-like notation, this request is stated as *SELECT * FROM sensors WHERE id IN (2,3) SAMPLE RATE 1s FOR 10s*. The laptop passes this query to node 1, which in turn forwards it to nodes 2 and 3. There, the query is executed and the results are returned.

This example illustrates three important aspects of middleware components in this category. Firstly, issued queries should be routed to the correct nodes and thus the middleware component should maintain some structured representation of the network. Secondly, the component needs a sensor-specific query language, which usually is adapted from the SQL language used to query conventional databases. In the query above, the keywords "SAMPLE RATE" and "FOR" are examples of keywords that have been added to the traditional SQL syntax. Thirdly, in conventional database systems the results of the queries are always immediately returned after it has been processed. In the context of sensor networks, however, queries can run and produce output continuously, resulting in a stream of data. Middleware components in this category include TINYDB [98], IRISNET [55], SINA [138], COUGAR [160], DSWARE [92], SNEE [50], and KSPOT [20].

Virtual Machine-inspired components. A Virtual Machine (VM) is a platform-independent programming environment that hides details of the underlying operating system and hardware. Software developers can thus write programs in one language, and deploy it to any device running a virtual machine. Since sensor nodes are based on a wide variety of software and hardware, using a virtual machine is also appropriate for sensor networks. When each sensor node runs a virtual machine, creating reusable programs for sensor networks becomes considerably easier. Figure 2.3 illustrates the setup of a typical VM-motivated component, containing a simple sensor network of three nodes with each of them running a virtual machine. The application on the laptop sends a program into the sensor network, where it arrives at node 1. There, it

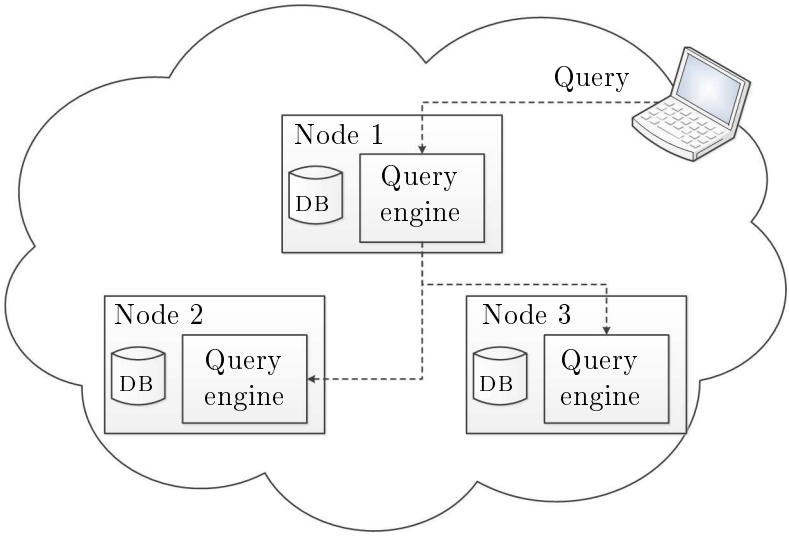


Figure 2.2: Simple sensor network where the nodes are viewed as parts of a distributed database. Each node comprises of a local database and query engine. The engine interprets and forwards queries through the network to the correct nodes, and executes them to retrieve data from its local database.

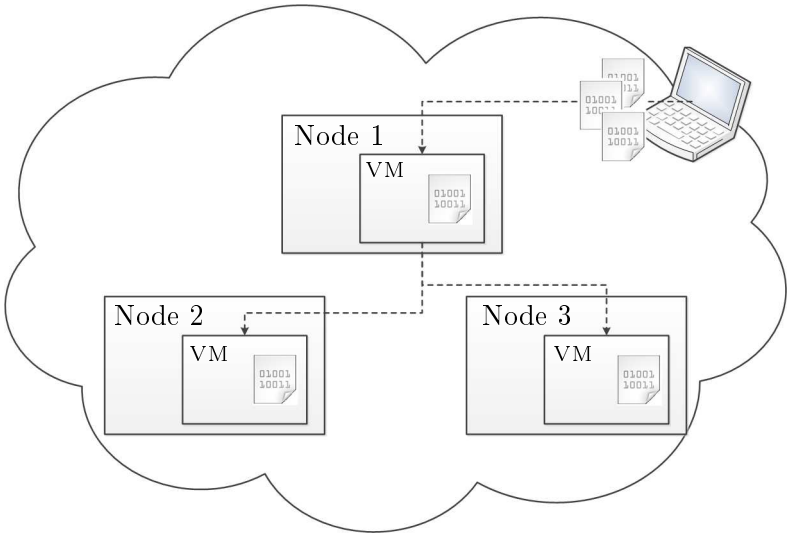


Figure 2.3: VM-based components deploy a Virtual Machine on each node, and thus provide an execution environment for small pieces of code. In this example, a program is split into three parts and forwarded to the appropriate node. There, the program is executed until it is finished.

is split into three parts and forwarded to the correct nodes. Each node then executes the part it receives, until the program ends. Applications for sensor networks typically use data from multiple sensor nodes, so a key feature of VM-motivated middleware components is that it facilitates the splitting of programs. Additionally, it sends the parts of the program to the correct sensor nodes, so the middleware component also needs to maintain a structured view of the network. Finally, the middleware component must manage software updates across the sensor network. MATÉ [90], MUSE [134], and MAGNET [94] are components in this category.

Agent-based components. Agents are small pieces of software that work together to achieve a predefined goal. Unlike conventional computer programs, agents are not activated by external commands but act autonomously based on a set of rules and on information from their environment. Additionally, agents are mobile and capable of moving from one environment to another. The principles of agent-based systems have also been applied in the context of sensor networks. To facilitate the execution and migration of agents in sensor networks, middleware components in this category typically equip each node in the network with a special execution environment (EE). This is illustrated in Figure 2.4, which represents a sensor network with three nodes and two agents, one at the second sensor node, and one en-route from node 1 to node 3. As with the VM-oriented components, agent-based middleware components need a structured view of the network, require special skills from the software developers, and must facilitate the distribution of software updates. Well-known agent-based approaches are IMPALA [95], AGILLA [49], SWAP [112], and MAPS [17].

Application-driven components. Whereas the middleware components in the previous categories were grouped by an architectural similarity, the components in this category have a common goal: optimizing Quality of Service (QoS). QoS aspects are typically application-specific, so this category is named 'application-driven'. An example of an application-driven middleware component is MILAN (Middleware Linking Applications and Networks) [65], which considers both the QoS requirements of an application and the QoS capabilities of the available sensor nodes. These requirements and capabilities are then matched to select the sensor nodes involved in the application. In MILAN, the QoS levels reflect uncertainty in sensor measurements, i.e., if faced with a choice between using a sensor node with level 0.7 and a node with level 0.8, the latter is preferred due to its higher reliability. Besides MILAN, MIDFUSION [19] and the Adaptive Middleware Component (AMC) from [72] also belong to this category.

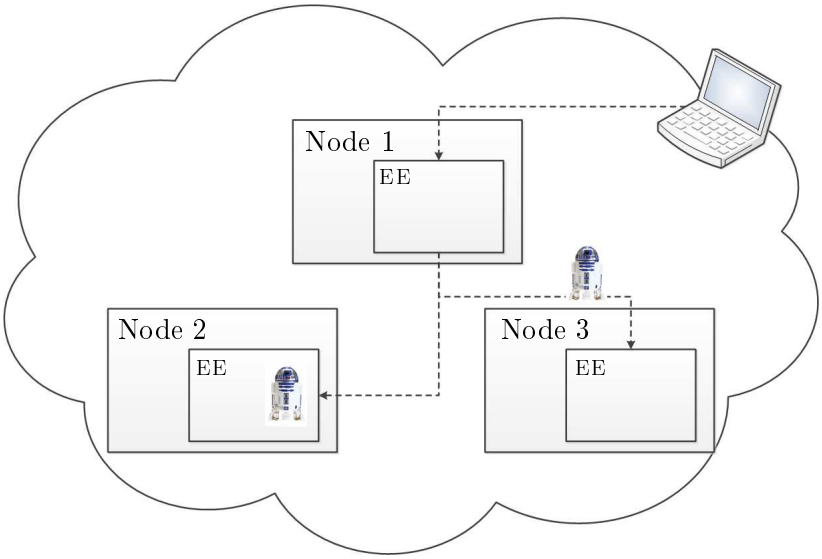


Figure 2.4: Agent-based middleware components equip each sensor node with an execution environment (EE) allowing agents to, e.g., migrate to other sensors.

Message-oriented components. In sensor networks, measurements often occur as a result of events that happen in the monitored environment. For instance, a high temperature measurement might trigger the transmission of that measurement to all interested applications. In traditional IT-systems, the *Publish/Subscribe* mechanism is often used to provide such event-driven communication, and it has also been applied in the context of sensor networks. In the Publish/Subscribe mechanism each sensor node broadcasts a standardized description of its capabilities across the network to interested applications. If an application is interested in the measurements of a sensor, it notifies the corresponding sensor node that it wants to subscribe to the sensor's events. When a sensor node detects an event, the resulting measurement is forwarded ('published') across the network to all subscribing applications. This mechanism ensures that subscribers are notified when an event occurs.

We use Mires from [145] as an example of a middleware component in this category, with Figure 2.5 illustrating the architecture of one single sensor node as used by Mires. At the bottom are the hardware components of the node, such as the sensors (measuring, e.g., temperature), the CPU, and the radio. When a new temperature measurement is done, the operating system reports it to the *Publish/Subscribe Service*. This service is at the heart of Mires,

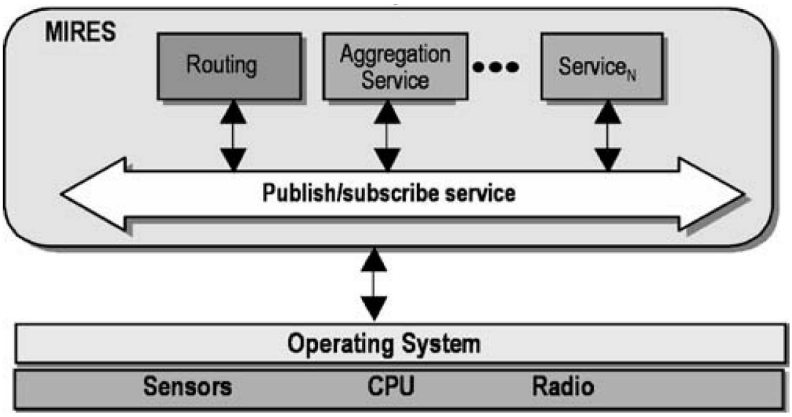


FIGURE 2.5: Architecture of one MIRES node, adapted from [145].

and provides communications between the service on the sensor node, and also to services on other nodes in the network. The measured temperature value can, e.g., be handed off to the (local) *Aggregation Service*, or it might be routed (via the *Routing Service*) to other sensor nodes. The *Publish/Subscribe* mechanism is flexible, so multiple services might be notified simultaneously of a new temperature value as well. Applications are built using MIRES by identifying what services are required from which node, and then subscribing to those services.

The event-based nature of many sensor networks makes the Publish/Subscribe mechanism a useful tool for middleware components in this category. The mechanism fully decouples applications and sensor nodes, so that adding or removing applications and/or sensor nodes does not require global changes to the middleware component. Also, the Publish/Subscribe mechanism hides vendor and hardware specific details about sensor nodes from applications. Thereby, developing applications in networks with heterogeneous sensor nodes becomes more convenient. Components AWARE [119], WMOS [162], and TINYMQ [140] also belong to this category.

2.3 Centralized middleware components

Most of the middleware components in the categories from the previous sections have not been widely adopted in practice. One of the reasons for this is that the middleware components were motivated by increased technological possibilities of sensor nodes (e.g., more memory, larger storage capacity, and

better processors), but many of the sensor nodes used in practice today are still resource-limited devices with no possibilities to run, e.g., a virtual machine. A second reason is that industry standards have received little attention in the scientific literature for sensor middleware components (e.g., Zigbee [166]). As the authors of [113] put it: *Academic WSN research and ZigBee appear to intersect only seldom, if at all. [...] This progressively caused industry to lose interest in academic* WSN *research, as compliance with standards [...] is key to industry applications.* Finally, one can question whether a sensor middleware component should extend into the sensor network. If it does, then the

Components	**Timeframe**	**Reference**	**Website**
Database-inspired			
SINA	2001	[138]	-
Cougar	2000-2005	[160]	[40]
IRISNet	2002-2005	[55]	[73]
TinyDB	2002-2005	[98]	[151]
DSWare	2004	[92]	-
KSPOT	2007-2011	[20]	[82]
SNEE	2008-2009	[50]	[144]
Virtual Machine-motivated			
Maté	2002-2005	[90]	[106]
Magnet	2001-2005	[94]	[99]
MUSE	2005	[134]	-
Agent-based			
Impala	2002-2004	[95]	-
Agilla	2004-2007	[49]	[15]
SWAP	2006	[112]	-
MAPS	2009-now	[17]	[101]
Application-driven			
MiLAN	2002-2004	[65]	-
AMC	2004	[72]	-
MidFusion	2008	[19]	-
Message-oriented			
Mires	2005	[145]	-
AWARE	2006-2009	[119]	[22]
WMOS	2011	[162]	-
TinyMQ	2011	[140]	-

Table 2.1: Overview of the middleware components per category. Each component has a reference to a paper describing the architecture, and a link to a website (if one exists).

middleware component is faced with typical network-issues such as routing, and perhaps the responsibility for this should be assigned to the network, not to the middleware component.

In recent years the scientific community has picked up on these concerns and included a new category of middleware components. Components in this category are separated from sensor technology and thus make no requirements in terms of, e.g., resources and access protocols. We call such components '*centralized middleware components*'. Figure 2.6 illustrates the role of a centralized middleware component schematically. It contains two sensor networks that are connected to a middleware component (at the bottom), and two applications that rely on sensor data (at the top). The middleware component forms a bridge between sensor technology and applications relying on sensor data, hence the term 'centralized'.

In the remainder of this section we describe several examples of centralized middleware components from the scientific literature, and discuss their advantages and disadvantages. Before that, in the next section, we describe the leading industry standard for sensor modeling, as this is included in many of the middleware components.

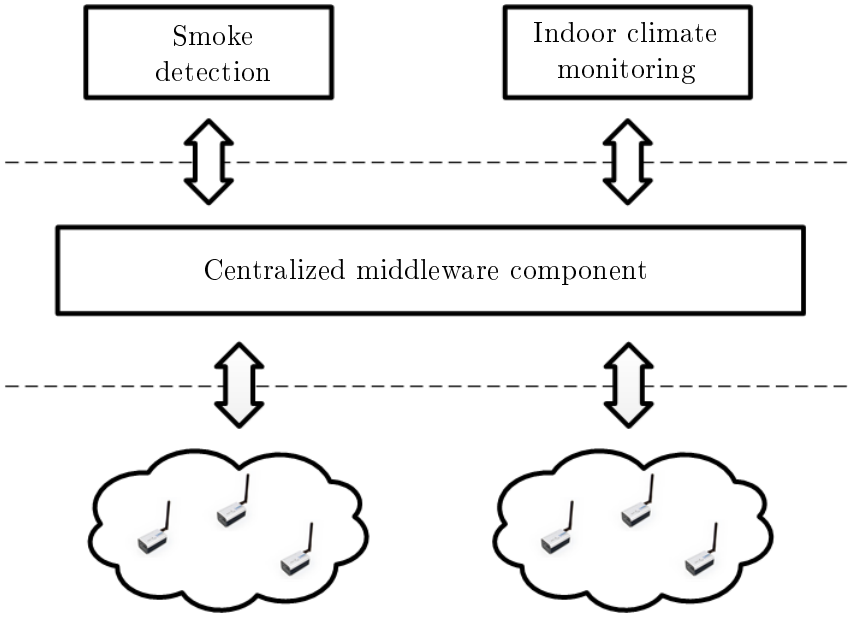


FIGURE 2.6: Position of the centralized middleware component with respect to sensor technology (at the bottom) and sensor applications (at the top).

2.3.1 Sensor Web Enablement initiative

Several years ago an international group of companies, government agencies and universities from the OpenGeospatial Consortium (OGC) [121] started the Sensor Web Enablement (SWE) initiative. This initiative aims to support the discovery and exchange of sensor information, as well as the tasking of sensor systems. It consists of standards covering the topics of modelling sensors and observations, and of interfaces for communicating with sensor nodes. The standards and interfaces in SWE are defined as web services, and include the following specifications:

- *Observations and Measurements*: a scheme for describing sensor observations and measurements.
- *Sensor Model Language*: an interface for describing sensor systems and their capabilities.
- *Sensor Observation Service* (SOS): a web service to obtain observations and sensor and platform descriptions from one or more sensor nodes.
- *Sensor Planning Service* (SPS): provides users with a standard interface for setting their own data collection requests.
- *Sensor Alert Service* (SAS): defines an interface for publishing and subscribing to sensor alerts.
- *Web Notification Service* (WNS): handles the asynchronous message delivery to the subscribers of the SAS and SPS.

Despite SWE's popularity, several drawbacks of standards have been identified in the scientific literature (from [23, 112]):

- There is no explicit ontological structure in the SWE framework.
- Security and privacy issues are not addressed.
- Conversion from a network-specific format to SWE standards requires detailed knowledge of both formats. Typically, off the shelf sensor nodes do not provide their data in SWE form. We return to this drawback in Section 2.4.
- There are no guidelines for the communication between services.
- Services are passive, so for example, a user can contact the SOS, but not vice versa.

We illustrate the SWE specifications with a use-case from the SensorSA middleware component [30], one of the components we describe in the following

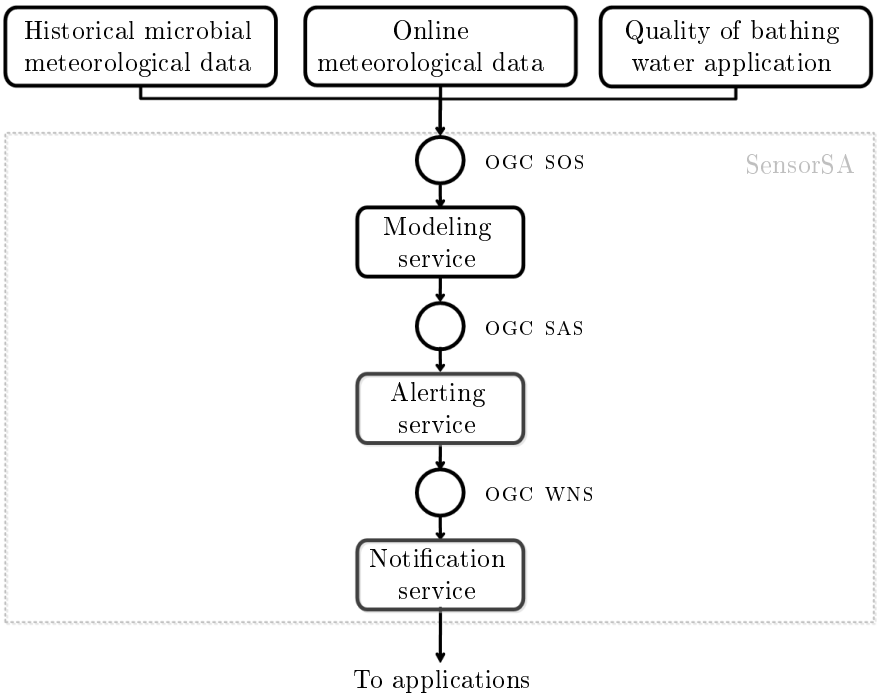


FIGURE 2.7: Abstract view of an application for monitoring the quality of seawater using the SENSORSA architecture, adapted from [30].

section. In this case study, a decision support system for marine risk management is created using SENSORSA. The system monitors the quality of seawater in areas where people often swim, and also predicts this quality for the near future. Authorities use this information to close beaches with a high risk of contamination, thereby preventing sickness due to microbial pollution.

An abstract view of the application is depicted in Figure 2.7. At the top, historical and online sensor data is imported into SENSORSA and used to update forecasting models. Importing is done using the *Sensor Observation Service*, which provides methods for requesting, filtering, and retrieving sensor data and information. The forecasting in this application is done by a (application-specific) modelling service, which generates an alarm when a prediction indicates bad water quality. This alarm is passed to the Alerting Service, an instance of the *Sensor Alert Service*. This service uses the Publish/Subscribe mechanism (which we saw earlier in Section 2.2 when discussing MIRES) to notify applications of alarms. Sending the alarm is done using *Web Notification Service*, which provides the ability to send an alarm as, e.g., an e-mail or a text message.

2.3.2 Overview of components

In this section we describe four centralized middleware components from the scientific literature. First, we describe SensorSA, because it is a good illustration of the SWE standards, and because of its clear documentation. Then we discuss SenseWeb, to demonstrate that non-SWE web services can also play a valuable role. Next, we present PULSENet, the most complete centralized middleware component available, featuring both SWE-based interfaces as well as various other industry standards for dealing with sensor data. Finally, we describe the middleware component LSM that utilizes the streaming nature of sensor data (one of the future research directions we identify in Section 2.4), and provides a way of publishing data without using web services. The four components are summarized in Table 2.2.

SensorSA

SensorSA (Sensor Service Architecture) is a middleware component developed in the SANY (Sensors Anywhere) project. SensorSA *aims to improve the interoperability of in-situ sensors and sensor networks, allowing quick and cost-efficient reuse of data and services from currently incompatible sources*[...] [30]. Its central role is illustrated in Figure 2.8, with in-situ sensors at the bottom, users at the top, and SensorSA in the middle.

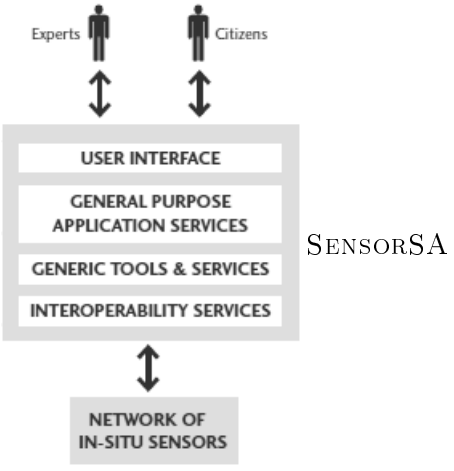


Figure 2.8: Illustration of the role of SensorSA, a centralized middleware component (adapted from [30]).

Its use of open standards from SWE makes SENSORSA an interesting component for companies seeking to include sensor data into their IT infrastructure. Several use cases are discussed in [30] and illustrate the use of SWE in practice. Additionally, SENSORSA uses several non-SWE interfaces from OGC for, e.g., visualization. SENSORSA also serves as an example of how a centralized approach simplifies security issues. Since the middleware component is not responsible for security on the sensor network (this network is considered to be owned and managed by a third party), it only needs to secure its own services. For this, SENSORSA relies on well-known security mechanisms for access control to service networks [30, Chap. 5]. SENSORSA contains several data fusion algorithms for analysis along both the time-dimension and the space-dimension of sensor data. Together with a time series toolbox for analyzing streaming sensor data, SENSORSA thus addresses two fields that we recognize as important future research directions in Section 2.4.

Despite the steps forward provided by SENSORSA, several of the drawbacks to SWE remain: there is no ontological structure, and services still seem to be passive. Moreover, there is no implementation of SENSORSA available for download, so a quick experiment with SENSORSA on an existing sensor network is not possible.

SENSEWEB

SENSEWEB is a sensor middleware component from Microsoft Research, *designed to let multiple concurrent applications share sensing resources contributed by several entities in a flexible but uniform manner* [76]. The key elements of SENSEWEB are illustrated in Figure 2.9, and strongly resemble the high-level description of a centralized middleware component from Figure 2.6. The primary building block is the *Coordinator*, which collects data via the *Sense gateway* and publishes this data to *Applications* and *Transformers*. Sensor nodes can be addressed through the Sense gateway, which provides a uniform interface for the rest of SENSEWEB, hiding any vendor-specific aspects. Transformers are components that process sensor data into other formats, for instance by calculating averages, or by creating figurative representations of data. In this way, transformers provide low-level elements that can be easily included in applications. The Coordinator consists of two separate modules, the *SenseDB* and the *Tasking Module*. SenseDB provides load-balancing facilities by analyzing requests for data to find overlap in their desired responses, and by using a cache for sensor data. Additionally, it is responsible for keeping track of the various sensors attached to the coordinator, and of their descriptions and ca-

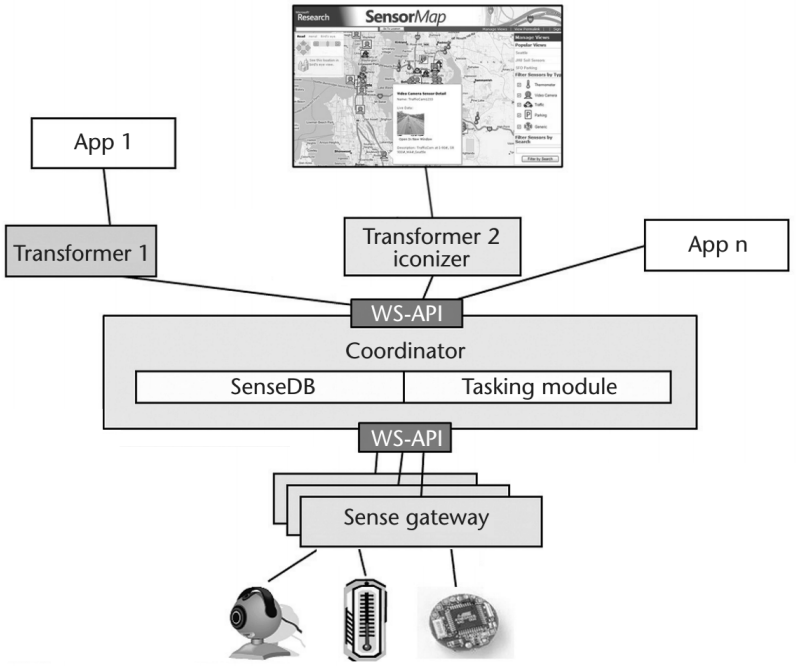


Figure 2.9: Overview of SenseWeb, illustrating how the coordinator mitigates between applications and sensor nodes (adapted from [76]).

pabilities. The Tasking Module determines which sensors are most suitable for answering a query, taking into account, e.g., bandwidth, availability, and power levels. Thus, these two models together provide an intelligent mechanism for load balancing, which is the key distinguishing feature of SenseWeb.

PULSENet

PULSENet is a sensor web component developed at the Northrop Grumman Corporation, with the objective to *provide a standards-based framework for the discovery, access, use and control of heterogeneous sensors, their metadata, and their observation data* [46]. It is based on the standards provided by SWE, supplemented by a wide variety of non-SWE standards for, e.g., describing public safety alerts, detailing military events, and visualization. Sensor networks are connected to PULSENet via plugins, which hide vendor-specific interfaces and perform the translation to and from the SWE standards. From a practical point of view, PULSENet has been tried and tested extensively. It is used, for instance, in the Defense and Intelligence domain, which contains sen-

sors and platforms with many modalities, levels of complexity, data formats, and privacy issues. Other domains include the Ocean Science community, and Air Quality applications. This emphasizes the practical relevance of PULSENet, and of centralized middleware components in general.

The authors of [46] also provide a list of best practices when dealing with SWE. These can be summarized as:

- *Apply the* SWE *standard only when needed.* Using SWE for data publishing typically means sacrificing some performance, due to SWE's complexities. So apply SWE only when a device has sufficient capacity to run web services and parse XML. Otherwise, consider using more low-level standards.
- *Keep it simple.* SWE is a large and flexible standard, offering both simple and complex data structures. Use the complex structures only if necessary to keep SWE overhead as low as possible.
- *Use [...] the* SWE *compliance tests.* OGC offers a compliance engine that allows third parties to test their implementations of SWE standards. Passing the compliance test adds significant value to a SWE-enabled middleware component.
- *Avoid reinventing the wheel.* Several open source implementations for SWE web services are available, and using them is advisable considering development time and software quality.

Unfortunately, the source code for PULSENet is unavailable from the corresponding website, so real-life experimentation with PULSENet is not possible. Also, we could not find a more detailed description of the PULSENet architecture than the one in [46].

LSM

LSM (Linked Stream Middleware) [87, 88] is a middleware component from the field of Linked Stream Data. It aims to simplify the integration of sensor data with data from other sources by providing semantic descriptions for sensor measurements and sensor data streams. As the name suggests, Linked Stream Data has two main properties: the data has mutual relationships (i.e., it is linked), and it is available via streams. The links are visible in Figure 2.10 in the Linked Data layer, and together form a complex structure of current and historical information. The data (both static and streaming) is collected in the

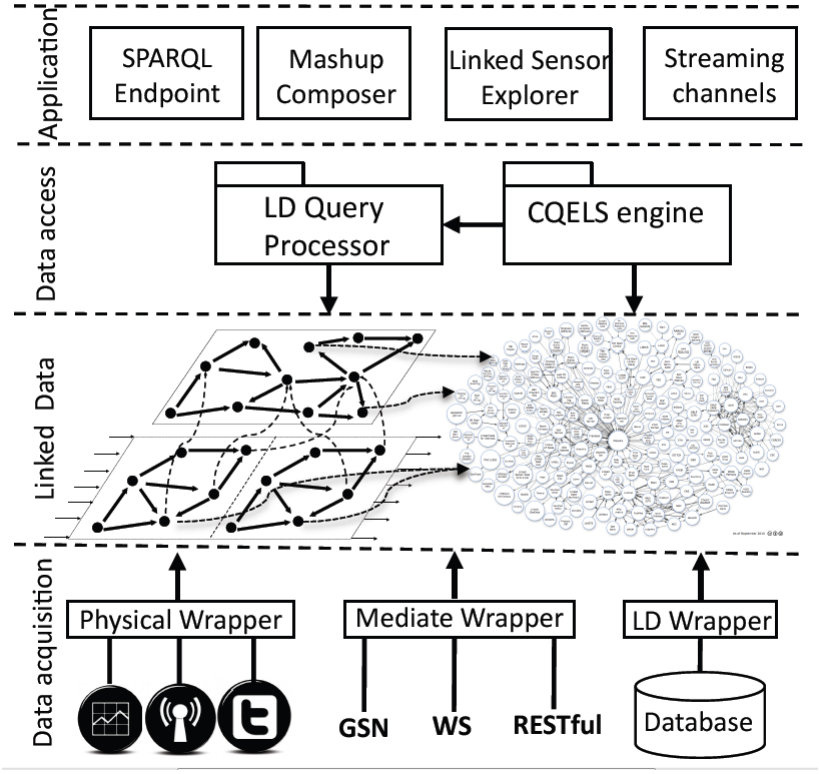


Figure 2.10: Overview of the elements of LSM, adapted from [87].

Data Acquisition layer, and transformed to a Linked Data format via 'Wrappers' (which are similar to, e.g., the plugins in PULSENet). Access to the data for applications is provided by a query processor, using a query language for streaming data: CQELS. This query language is not a standard, but developed by LSM's author in [86].

A nice feature of LSM is that it uses W3C's Semantic Sensor Network ontology [38], which also yields the relationships between data points (for instance, they can be linked via their 'location' property). By using an ontology, standard query options become available via SPARQL [128]. The query language CQELS is based on SPARQL and enables the expressiveness of an ontology for streaming data. A working demo of LSM is available online at `http://lsm.deri.ie/`. Although the concept of Linked Stream Data is promising, it is relatively new in the context of sensor networks, and therefore untested in practice. More research and experiments are necessary to demonstrate if, e.g., the CQELS query language is applicable in a broad range of applications.

Component	Timeframe	Reference	Website
SENSORSA	2006-2010	[30]	-
SENSEWEB	2006-2010	[76]	[135]
PULSENET	2009	[46]	[129]
LSM	2011-2012	[87, 88]	[96]

TABLE 2.2: Overview of four centralized middleware components described in this chapter.

2.4 Directions for future research

Centralized middleware components form a bridge between sensor technology and applications relying on sensor data. Since sensor technology is based on a wide variety of standards and protocols, a centralized middleware component should support many different technologies. The components discussed in Section 2.3 recognize this need, and provide gateways (SENSEWEB), plugins (SENSORSA, PULSENET), or wrappers (LSM) for implementing support for various technologies. Similarly, interaction with applications occurs via various different (web-)interfaces, and middleware components should support these as well. The paper describing PULSENET [46] nicely illustrates the diversity in web interfaces. From the point of view of the centralized middleware component, the large diversity of technologies can be seen as a nuisance, since it causes a lot of extra work. But support for many different technologies makes it considerably easier to develop sensor-driven applications, and is essential for the continued growth of, e.g., The Internet of Things.

A next step for centralized middleware component research concerns the 'classic' issues of Quality of Service, privacy, and security. Much research on these topics already exists (see, e.g., [36], [105], and [133, 165], respectively), but has been hampered by a lack of clear definitions in the context of sensor networks. As applications relying on sensors and sensor networks become ubiquitous, research on Quality of Service, privacy, and security will most likely regain momentum.

A third promising research direction is formed by semantic specifications of sensors. Giving a semantic description of a sensor makes it clear what type a sensor is (e.g., a 'Temperature' sensor), what units its measurements are in (e.g., 'Degrees Celsius'), and how these measurements were obtained (e.g., 'Average of 10 measurements in the last 1 second'). Such properties of sensors become particularly important once sensor data is used by third party applica-

tions, because they must understand exactly what the offered data represents. A good overview of this topic is presented in [37, 38].

Furthermore, as evidenced by LSM, techniques from the 'Data Streams' domain will become more popular. A large body of literature is available (see, e.g., [14, 51]) and is ready to be applied. Of particular interest are applications that combine streaming sensor measurements with static data from other sources, because centralized middleware components are in a unique position to collect and process both types of data.

We foresee that applications will increasingly combine sensor data with other sources of data, driven by newly available sensor data from middleware components. For instance, information from CO_2 and temperature sensor nodes in a building can be combined with data from security systems to verify alarms. If a security system gives an alarm that a burglary is in progress, an unexpected change in CO_2 and temperature measurements might confirm that something irregular is happening. This is an example of a process known as 'Data Fusion', and we think that techniques from this domain can boost the development of a new generation of innovative and intelligent sensor-related applications. Interested readers are referred to [77], which contains a review of the state-of-the-art in this domain.

2.5 Conclusion

This chapter reviewed middleware components for sensor networks in the literature using a well-known categorization. Then, we described that recently, a new category of sensor network middleware components has emerged. These components do not run part of the middleware component on the sensor nodes, contrary to many existing components. We introduced the term 'centralized middleware component' for this new type of component and discussed four examples of this type. Finally, we discussed directions for future research.

3

Outlier Preservation by Dimensionality Reduction Techniques

In the previous chapter we discussed how middleware components bridge the gap between sensor networks and applications that rely on the data produced by these networks. With sensors playing an increasing role in technologies and in our lives, applications can choose many types and sources of data that are available at middleware platforms. How can all this information be transformed into actionable insight? Providing a short insightful summary that helps users identify events and take appropriate action is essential. Inevitably though, some information is always lost when providing a summary, so the technique used to create it should be chosen carefully. In this chapter we focus our attention on *Dimensionality Reduction* (DR), a family of techniques often used for creating short summaries. We study the effect of such techniques on outliers – measurements in the data that do not conform to regular patterns. We demonstrate that dimensionality reduction can indeed have a large impact on outliers. To that end we apply three dimensionality reduction techniques to three real-world data sets, and inspect how well they preserve outliers. We use several performance measures to demonstrate how well these techniques are capable of preserving outliers, and we discuss the results.

This chapter is based on the results presented in [4].

3.1 Introduction

Recent technological developments have resulted in a broad range of cheap and powerful sensor nodes, enabling companies to use sensor networks in a cost-effective way. Sensor networks will increasingly become part of our daily life – envision, e.g., a house with sensors related to smoke detection, lighting control, motion detection, environmental information, security issues, and structural monitoring. Combining all this information to actionable insights is a challenging problem. For instance, in the event of a burglary in a house, the sensors involved in motion detection, environmental monitoring, and security all yield useful information. Providing a short insightful summary that helps users identify the event and take appropriate action is essential. Dimensionality reduction is a family of techniques aimed at reducing the number of variables (dimensions) in the data and thus at making the data set smaller. In essence, it helps identify what is important, and what is not.

Dimensionality reduction often results in some loss of information, and applications might be affected by this loss. For instance, the burglary mentioned before is a (hopefully) rare event that is different from normal patterns in the sensor data (i.e., a so-called *outlier*). Unfortunately, DR-methods often lose outliers among the regular sensor data. Figure 3.1 illustrates this situation using a two-dimensional data set with an outlier near the top-left corner. When dimensionality is reduced by projecting all points onto a line, the outlier is mapped into the center of the reduced data set (the middle arrow in Figure 3.1), and is thus no longer an outlier. So dimensionality reduction might lose outliers among regular points, causing problems for applications relying on the detection of outliers.

A solution to this problem is to identify outliers prior to applying DR. This is, however, not always computationally feasible due to the high dimensionality of the data, particularly when an outlier involves multiple dimensions. The point in the top-left corner of Figure 3.1 is an example of such an outlier: it is not an outlier in either the x- or y-dimension, but clearly is an outlier in the (x, y)-plane. In such a computationally challenging situation, it might be more efficient to apply DR first, followed by the detection of outliers.

Motivated by this, in this chapter we experimentally determine how well DR-techniques preserve outliers. To this end, we describe three well-known DR-techniques that are relevant for a broad audience, and apply them to several real-world data sets from a sensor-related context. For each DR-technique we capture its capability to preserve outliers in three performance measures, and

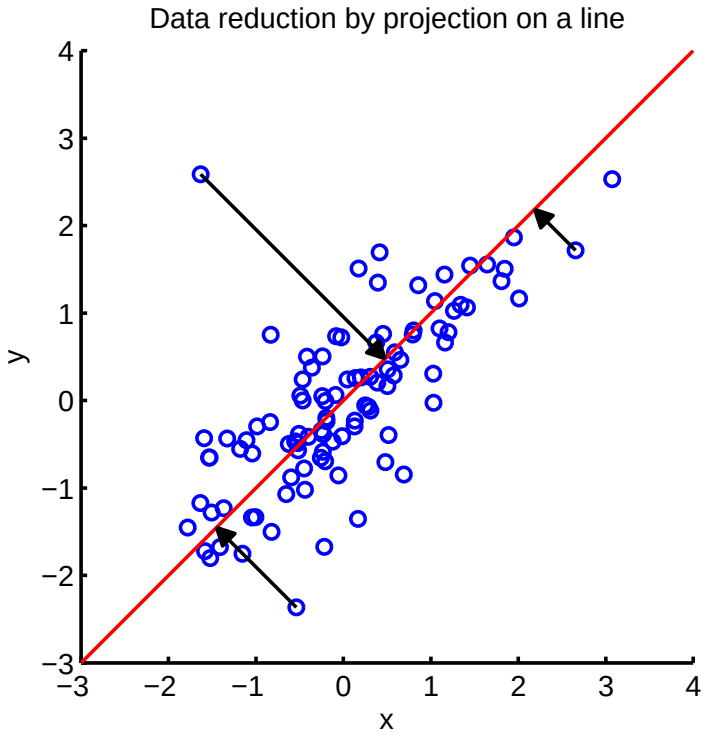


Figure 3.1: A two-dimensional data set reduced to one dimension, with an outlier (middle arrow) mapped to the center of the reduced data set.

compare the results. From the three techniques we identify the one with the best performance, and discuss the intuitions behind the scores.

A large body of literature exists on dimensionality reduction, and an overview of techniques from this field can be obtained from [33, 48, 59, 78, 120, 157]. DR-techniques are typically used for visualization [91, 152], as a preprocessing step for further analysis [52, 131, 153], and for increasing computational efficiency [42, 62]. Outlier detection is a popular research topic as well, and is comprehensively reviewed in survey papers [16, 70, 97, 163, 164]. Certain specific topics, such as intrusion detection [56] and fraud detection [26, 124], are closely related to outlier detection. In [114] the authors consider Kohonen's Self-Organizing Maps (SOM, [79]) and how this DR-technique can be used to identify outliers. [64] illustrates the effect of outlier-removal on Isomap [150], another DR-technique. [34] looks at local DR, where reduction is applied to previously identified clusters. Outlier detection occurs as part of the cluster-identification phase.

These papers do not, however, look at outlier *preservation* by DR-techniques, as discussed in this chapter. In [45], the authors compare multiple outlier detection methods on various data sets, including one data set with its dimensionality reduced. As in this chapter, their analysis also suggests that outlier detection is affected by dimensionality reduction, although they only use one DR-method and one performance measure. In [115], a setup is used that is close to our approach: four DR-methods are applied to three data sets, and the performance (using one score measure) is inspected for two outlier detection methods. However, the DR-methods in [115] are selected from the feature extraction domain, and are not well-known in the DR-community.

The structure of this chapter is as follows: Section 3.2 describes the DR-techniques, Section 3.3 contains the outlier detection method as well as the performance measures. Then, in Section 3.4 we describe the data sets that we use in the experiments. Section 3.5 demonstrates the output of the experiments and discusses the results, followed by conclusions, recommendations, and ideas for further research in Section 3.6.

3.2 Dimensionality reduction techniques

Denote by n the number of measurements and by d the number of sensors producing the measurements. The number of sensors is known as the *dimension* of the data, and DR-techniques aim to lower this dimension to a smaller value. More formally, if the measurements are vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, then DR-techniques try to find points $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^{d'}$ with $d' < d$. This section describes three well-known and often used DR-techniques: *Principal Component Analysis* (PCA), *Multidimensional Scaling* (MDS), and *t-Stochastic Neighbourhood Embedding* (t-SNE).

3.2.1 Principal Component Analysis

Principal Component Analysis was initially proposed in [122]. It finds a low-dimensional representation of the data with minimal loss of variation of the data set. Suppose that we have n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ (corresponding to n measurements from each of d sensors in this chapter), and that they are placed in the $n \times d$ matrix X. We denote the $d \times d$ correlation matrix of X by C, its eigenvalues by $\lambda_1, \ldots, \lambda_d$, and its eigenvectors by $\mathbf{u}_1, \ldots, \mathbf{u}_d \in \mathbb{R}^d$. Typically, the eigenvalues are ordered s.t. $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$, and the

eigenvectors are orthogonal to each other. The eigenvalues reflect the amount of variance in the data set explained by the corresponding eigenvectors. To be precise, the first d' eigenvalues explain a fraction $\left(\sum_{k=1}^{d'} \lambda_k\right)/\left(\sum_{k=1}^{d} \lambda_k\right)$ of the variance.

PCA achieves dimensionality reduction by omitting eigenvectors $\mathbf{u}_{d'+1}, \ldots \mathbf{u}_d$, with d' the smallest integer such that the fraction of explained variance exceeds a threshold $\tau \in [0, 1]$. This threshold is a parameter of PCA. Summarized, the process works as follows:

1. Construct the data matrix X.
2. Compute the correlation matrix C.
3. Find the n eigenvalues λ_k and eigenvectors $\mathbf{u}_k$ of C.
4. Determine d' such that $\left(\sum_{k=1}^{d'} \lambda_k\right)/\left(\sum_{k=1}^{d} \lambda_k\right) < \tau$.
5. Construct matrix $\hat{U} = [\mathbf{u}_1 \ldots \mathbf{u}_{d'}]$.
6. Reduce dimension by computing $\hat{X} = X\hat{U}^T$.

The $n \times d'$ matrix $\hat{X}$ matrix contains n data points $\hat{\mathbf{x}}_\mathbf{1}, \ldots, \hat{\mathbf{x}}_\mathbf{n} \in \mathbb{R}^{d'}$ that form the reduced data set. The vectors $\mathbf{u}_k$ $(1 \leq k \leq d)$ vectors are called the *Principal Components*, and give PCA its name. A more detailed description and examples of PCA can be found in, for instance, [63, 74, 84, 149].

3.2.2 Multidimensional Scaling

Multidimensional Scaling is the name of a family of dimensionality reduction techniques that preserve distances in the data set. The *classical* version of MDS finds points $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^{d'}$ in a low-dimensional space that minimize

$$\min_{\mathbf{y}_1,\ldots,\mathbf{y}_n} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(||\mathbf{x}_i - \mathbf{x}_j|| - ||\mathbf{y}_i - \mathbf{y}_j||\right)^2. \tag{3.1}$$

Here $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ are the high-dimensional points, and $||\cdot||$ is the Euclidean distance in the respective space. The classical version of MDS is equivalent to PCA, see for instance [54]. Other members of the MDS family use a different distance measure or a different quantity to optimize than Eq. (3.1). We use a version of MDS with the so-called *squared stress* criterion

$$\min_{\mathbf{y}_1,\ldots,\mathbf{y}_n} \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} \left(||\mathbf{x}_i - \mathbf{x}_j||^2 - ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^2}{\sum_{i=1}^{n} \sum_{j=1}^{n} ||\mathbf{x}_i - \mathbf{x}_j||^4}. \tag{3.2}$$

For the distance measure $||\mathbf{x}_i - \mathbf{x}_j||$ we do not use the Euclidean distance measure as in the classical version of MDS. To see why, note that MDS with the Euclidean distance is sensitive to natural variations in the data. Consider, for instance, a data set consisting of two columns, one with values uniformly drawn from the interval $[1000, 2000]$ and one with values drawn from $[0, 1]$. Clearly, all values in the first column are several orders of magnitude larger than those in the second column. When minimizing the quantity in Eq. (3.1) the procedure focuses on the elements of the first column, since that brings it closest to the minimum. In essence, the second column is ignored and MDS is biased towards the first column.

To overcome this problem, the Euclidean distance is typically replaced by the *Mahalanobis* distance [100]:

$$||x_i - x_j||_M = \sqrt{(\mathbf{x_i} - \mathbf{x_j})\Sigma^{-1}(\mathbf{x_i} - \mathbf{x_j})^T}, \tag{3.3}$$

where Σ is the covariance matrix. By including the covariance matrix in the distance measure, the natural variations in the data are removed and thus MDS is unbiased with respect to dimensions. Eq. (3.1) then becomes

$$\min_{\mathbf{y}_1,\ldots,\mathbf{y}_n} \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(||\mathbf{x}_i - \mathbf{x}_j||_M^2 - ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^2}{\sum_{i=1}^{n}\sum_{j=1}^{n}||\mathbf{x}_i - \mathbf{x}_j||_M^4}. \tag{3.4}$$

Note that the Mahalanobis distance is only used for the high-dimension points $\mathbf{x}_i$, because the low-dimensional points $\mathbf{y}_i$ are found by the minimization.

3.2.3 t-Stochastic Neighbourhood Embedding

Stochastic Neighbourhood Embedding

t-Stochastic Neighbourhood Embedding is a variation on *Stochastic Neighbourhood Embedding* (SNE), first proposed in [68]. SNE presents the novel idea of defining a probability that two points are neighbours. If the distance between two points is small, SNE assigns a high 'probability of being a neighbour' to this pair. Similarly, points that are far apart are assigned a low 'probability of being a neighbour'. SNE reduces dimensionality by looking for low-dimensional points that preserve the assigned probabilities.

In SNE, the probability assigned to two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is

$$p_{i|j} = \frac{e^{-||\mathbf{x}_i - \mathbf{x}_j||_M^2/2\sigma_i^2}}{\sum_{k=1,k\neq i}^{n} e^{-||\mathbf{x}_i - \mathbf{x}_k||_M^2/2\sigma_i^2}}. \tag{3.5}$$

The parameter σ_i is set by hand or determined with a special search algorithm. Note how we again employ the Mahalanobis distance from Eq. (3.3) for the high-dimensional points in Eq. (3.5). Also, observe that points that are close together result in a large value for $p_{i|j}$, and that points that are far away from each other yield a low value for $p_{i|j}$.

In low-dimensional space, probabilities similar to those in Eq. (3.5) are defined as

$$q_{i|j} = \frac{e^{-||\mathbf{y}_i - \mathbf{y}_j||^2}}{\sum_{k=1, k\neq i}^{n} e^{-||\mathbf{y}_i - \mathbf{y}_k||^2}}. \tag{3.6}$$

The parameter σ_i is not necessary here, because it would only lead to a rescaling of the resulting low-dimensional points $\mathbf{y}_i$. The $\mathbf{y}_i$ are then found by minimizing the Kullback-Leibler divergence of these two probability distributions

$$\min_{\mathbf{y}_1,\ldots,\mathbf{y}_n} \sum_{i=1}^{n}\sum_{j=1}^{n} p_{i|j} \log \frac{p_{i|j}}{q_{i|j}}. \tag{3.7}$$

Minimization of Eq. (3.7) can be done with, e.g., the gradient descent algorithm, or the scaled conjugate gradients procedure.

t-SNE

In [156] the authors propose t-SNE, which differs from SNE in two aspects. First, note that the probabilities in Eq. (3.5) are not necessarily symmetric, i.e., $p_{i|j}$ and $p_{j|i}$ do not need to be equal. This complicates minimization of Eq. (3.7), because it has twice as many variables as in the symmetric case. In t-SNE, the $p_{i|j}$ in Eq. (3.7) are replaced by p_{ij}:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n},$$

with $p_{i|j}$ still computed from Eq. (3.5). Note that p_{ij} is symmetric in i and j, and thus reduces the number of variables in the minimization of the Kullback-Leibler divergence by a factor two. Additionally, this change ensures that $\sum_{j=1}^{n} p_{ij} > 1/(2n)$ so that each point (including outliers) has a significant contribution to the cost function.

The second change proposed for t-SNE concerns the q_{ij}. Instead of using Gaussian-style probabilities as in Eq. (3.6), t-SNE uses probabilities inspired

by the Student t-distribution (with one degree of freedom):

$$q_{ij} = \frac{(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2)^{-1}}{\sum_{k=1,k\neq i}^{n}(1 + ||\mathbf{y}_i - \mathbf{y}_k||^2)^{-1}}.$$

This distribution has heavier tails than the Gaussian used by SNE, so it maps nearby high-dimensional points less nearby in low-dimensional space than SNE. A justification for this approach comes from the so-called *Crowding problem*: there is much more room in high-dimensional space for points, so in a low-dimensional representation data points tend to be 'squeezed' together. By using the Student t-distribution, these crowded points are placed just a bit further apart.

Low-dimensional points are still found by optimizing the Kullback-Leibler divergence from Eq. (3.7), but with $p_{i|j}$ replaced by p_{ij} and $q_{i|j}$ by q_{ij}:

$$\min_{\mathbf{y}_1,\ldots,\mathbf{y}_n} \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{3.8}$$

3.3 Experimental setup

We adopt the following experimental setup when investigating dimensionality reduction for outlier preservation:

1. Normalize each data set such that it has zero mean and unit variance. This is a common preprocessing step for experimental data.
2. Find outliers in the high-dimensional (centered and scaled) data set.
3. Reduce the data set to two dimensions.
4. Again look for outliers, this time in the low-dimensional data.
5. Compute a score reflecting the performance of each DR-method on the data set.

We apply this setup to the DR-techniques from Section 3.2 and to a number of real-world data sets, described later in Section 3.4. Prior to that, the sections below describe the technique that we use for outlier detection, and three performance measures that we use to assess how well outliers are preserved. For the DR-techniques we used Matlab implementations available in the *Dimensionality Reduction Toolbox* [155].

Algorithm 3.1 Peeling

1. Calculate the convex hull around all the points in the data set.
2. Find the point on the hull with the largest (Mahalanobis) distance to all other points in the data set.
3. Remember the outlier and remove it from the data set.
4. Calculate the new convex hull, and check if the stop criterion is reached. If so, stop, otherwise continue with step 2.

3.3.1 Onion peeling

The idea of *Onion Peeling*, or Peeling in short, is to construct a convex hull around all the points in the data set and then find the points that are on the convex hull. These points form the first 'peel' and are removed from the data set. Repeating the process gives more peels, each containing a number of points. This technique can be utilized for finding outliers, if we consider a point in the data set to be an outlier if they have a large distance to the other points in the data set. With this intuitive interpretation of an outlier, the largest outlier in the data set is on the first peel. By inspecting the total distance of each point on the hull to all other points in the data set, we can find the one with the largest total distance. Removing this point from the data set and repeating the process gives new outliers. The decrease in volume of the convex hull after removing an outlier is used as a stop criterion. Once the volume decreases by a fraction less than α ($0 \leq \alpha \leq 1$), we stop looking for outliers. In our experiments we set $\alpha = 0.005$. Although with this procedure there is no guarantee that all outliers are found, it is sufficient for the data sets in this chapter. Peeling is outlined in Algorithm 3.1.

3.3.2 Measuring performance

After running the experiment for one data set and one DR-method, we need to quantify the performance of this method with respect to the preservation of outliers. In order to do so, we assign each point to one of four groups:

- True Positive (TP). The point is an outlier both before and after DR.
- False Positive (FP). The point is not an outlier before DR, but is one after.
- False Negative (FN). The point is an outlier before DR, but not after.
- True Negative (TN). The point is not an outlier before DR, nor after.

		Outlier before DR?	
		Yes	**No**
After DR?	**Yes**	TP	FP
	No	FN	TN

FIGURE 3.2: Confusion matrix indicating what happened to outliers after DR.

We can summarize these quantities in a *confusion matrix*, as demonstrated in Figure 3.2. In an ideal scenario the confusion matrix would be diagonal (i.e., 0 FPs and FNs), indicating that all outliers and non-outliers were correctly retained by the DR-methods. However, in practice the matrix often contains some FPs and FNs, and the performance of a DR-method is judged by all four quantities. Confusion matrices are used in several research communities to assess the performance of, e.g., binary classifiers and statistical tests. Often a single number is needed to capture performance, which subsequently results in a combination of the four quantities in the table. Several such combinations exist and are used in various fields of research (see the overview in [127] for more information).

We describe three performance measures that are often used in the literature, but before we do so we highlight one complicating aspect of our problem scenario. Most practical data sets have a significantly larger number of non-outliers than outliers, so in the confusion matrix the TN is usually the largest number. As an example of a performance measure that is affected by this, we look at *accuracy*, defined as

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}.$$

Since TN is the dominating number in this expression, accuracy is always close to 1, making it difficult to identify small differences in performance. The three performance measures described below are selected because they are capable of handling this issue.

F1-score

The *F1-score* is a combination of *recall* and *precision*:

- Recall: the fraction of high-dimensional outliers that is retained by the DR-method (i.e., TP/(TP + FN)), which is maximized when FN equals 0.
- Precision: the fraction of low-dimensional outliers that were also high-dimensional outliers (i.e., TP/(TP + FP)), which is maximized when FP equals 0.

The F1-score takes the harmonic mean of precision and recall, resulting in a number between 0 (when TP=0) and 1 (when FP=FN=0):

$$\begin{aligned} F1 &= 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \\ &= 2 \cdot \frac{\text{TP}/(\text{TP}+\text{FP}) \cdot \text{TP}/(\text{TP}+\text{FN})}{\text{TP}/(\text{TP}+\text{FP}) + \text{TP}/(\text{TP}+\text{FN})} \\ &= \frac{2\text{TP}}{(2\text{TP} + \text{FN} + \text{FP})}. \end{aligned} \tag{3.9}$$

If TP + FN = 0 or TP + FP = 0 then the F1-score is defined as 0. Note that the element TN of the confusion table does not affect the score, and it is therefore not affected by the sparsity of outliers. The F1-score is used in, e.g., Information Retrieval [32, 103] and Machine Learning [45, 136, 154].

Matthews correlation

The *Matthews Correlation* [107] computes a correlation coefficient between the class labels (i.e., outlier or non-outlier) in high and low dimension of each point in the data sets. It results in a number between −1 (perfect anti-correlation) and 1 (perfect correlation), with 0 indicating the absence of correlation. Below we derive an expression for the Matthews Correlation in terms of the elements of the confusion matrix. Denote the class labels in low-dimensional space by $l_1 \cdots l_n$ and those in high-dimensional space by $h_1 \cdots h_n$, i.e.,

$$h_i = \begin{cases} 1 & \text{if point } i \text{ is an outlier in high dimension,} \\ 0 & \text{otherwise,} \end{cases}$$

and

$$l_i = \begin{cases} 1 & \text{if point } i \text{ is an outlier in low dimension,} \\ 0 & \text{otherwise.} \end{cases}$$

Here, n is still the total number of points in the data set. The Matthews correlation can be interpreted as a measure of how well outliers are preserved. It is denoted by ρ, and computed from

$$\rho = \frac{1}{n-1} \frac{\sum_{i=1}^{n}(l_i - \bar{l})(h_i - \bar{h})}{\sigma_l \sigma_h}, \tag{3.10}$$

where

$$\bar{l} = \frac{1}{n}\sum_{i=1}^{n} l_i = \frac{\text{TP} + \text{FP}}{n}, \qquad \bar{h} = \frac{1}{n}\sum_{i=1}^{n} h_i = \frac{\text{TP} + \text{FN}}{n}, \tag{3.11}$$

using notation from the confusion matrix. The σ_l is the standard deviation of the l_i, i.e.,

$$\begin{aligned}
\sigma_l &= \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(l_i - \bar{l})^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(l_i^2 - 2l_i\bar{l} + \bar{l}^2)} \\
&= \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(l_i - 2l_i\bar{l} + \bar{l}^2)} = \sqrt{\frac{n\bar{l} - 2n\bar{l}^2 + n\bar{l}^2}{n-1}} \\
&= \sqrt{\frac{n}{n-1}}\sqrt{\bar{l}(1-\bar{l})}.
\end{aligned}$$

Similarly, the standard deviation of the h_i becomes $\sigma_h = \sqrt{\frac{n}{n-1}}\sqrt{\bar{h}(1-\bar{h})}$. Substituting these quantities into Eq. (3.10) yields

$$\begin{aligned}
\rho &= \frac{\sum_{i=1}^{n}(l_i - \bar{l})(h_i - \bar{h})}{\sigma_l \sigma_h} = \frac{\sum_{i=1}^{n}(l_i - \bar{l})(h_i - \bar{h})}{n\sqrt{\bar{l}\bar{h}(1-\bar{l})(1-\bar{h})}} \\
&= \frac{\sum_{i=1}^{n}(l_i h_i - \bar{l}h_i - l_i\bar{h} + \bar{l}\bar{h})}{n\sqrt{\bar{l}\bar{h}(1-\bar{l})(1-\bar{h})}} = \frac{\sum_{i=1}^{n}(l_i h_i) - n\bar{l}\bar{h} - n\bar{l}\bar{h} + n\bar{l}\bar{h}}{n\sqrt{\bar{l}\bar{h}(1-\bar{l})(1-\bar{h})}} \\
&= \frac{\sum_{i=1}^{n}(l_i h_i) - n\bar{l}\bar{h}}{n\sqrt{\bar{l}\bar{h}(1-\bar{l})(1-\bar{h})}}.
\end{aligned}$$

Using $\sum_{i=1}^{n}(l_i h_i) = \text{TP}$ and Eq. (3.11), some algebra yields

$$\rho = \frac{\text{TP}\cdot\text{TN} - \text{FP}\cdot\text{FN}}{\sqrt{(\text{TP}+\text{FN})(\text{TP}+\text{FP})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}}. \tag{3.12}$$

If any of $\text{TP}+\text{FN}, \text{TP}+\text{FP}, \text{TN}+\text{FP}$, or $\text{TN}+\text{FN}$ are 0, then ρ is defined as 0. Note that, since ρ is a correlation, it is not affected by the large number of non-outliers. The Matthews Correlation is often used in Bioinformatics to assess the performance of classifiers, see, e.g., [75, 111, 139].

Relative information score

The Relative Information score was proposed in [80] and relies on ideas from the Information Theory field. In this section we derive an expression for the Relative Information score based on the confusion matrix. Suppose we consider one particular point, then a priori we can compute the probability that it is an outlier from the confusion matrix

$$\mathbb{P}(\text{outlier in high dimension}) = \frac{\text{TP} + \text{FN}}{n}.$$

After DR, we can compute this same probability for the same point as

$$\mathbb{P}(\text{outlier in low dimension} \mid \text{outlier in high dimension}) = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

The authors of [80] argue that any well-performing classifier (DR-method) should at least result in a confusion table with $\frac{\text{TP}}{\text{TP}+\text{FN}} > \frac{\text{TP}+\text{FN}}{n}$, otherwise it has lost information from the original data. This forms the basis for their Relative Information score.

We introduce some notation and denote by $\mathbb{P}(C_i = c)$ the probability that point i in the data set has class c, with $c = 1$ indicating that it is an outlier in high dimension, and $c = 0$ that it is a non-outlier. From the confusion matrix, we know that

$$\mathbb{P}(C_i = 1) = \frac{\text{TP} + \text{FN}}{n}, \tag{3.13}$$

$$\mathbb{P}(C_i = 0) = \frac{\text{FP} + \text{TN}}{n}. \tag{3.14}$$

After DR each point is again an outlier or non-outlier, but this time in low dimension. We denote the probability that point i in low dimension has class c, given that it also had class c in high dimension, by $\mathbb{P}(C_i' = c|C_i = c)$. From the confusion matrix, we find that

$$\mathbb{P}(C_i' = 1|C_i = 1) = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{3.15}$$

$$\mathbb{P}(C_i' = 0|C_i = 0) = \frac{\text{TN}}{\text{FP} + \text{TN}}. \tag{3.16}$$

In [80], the amount of information (as defined by [137]) necessary to correctly classify point i is measured as

$$-\log_2(\mathbb{P}(C_i' = c|C_i = c)).$$

A DR-method that satisfies $\mathbb{P}(C'_i = c|C_i = c) > \mathbb{P}(C_i = c)$ for point i then gets a positive score on point i of

$$\log_2\big(\mathbb{P}(C'_i = c|C_i = c)\big) - \log_2\big(\mathbb{P}(C_i = c)\big).$$

If $\mathbb{P}(C'_i = c|C_i = c) < \mathbb{P}(C_i = c)$ the score is

$$\log_2\big(1 - \mathbb{P}(C_i = c)\big) - \log_2\big(1 - \mathbb{P}(C'_i = c|C_i = c)\big),$$

which is negative. When $\mathbb{P}(C'_i = c|C_i = c) = \mathbb{P}(C_i = c)$ the score is defined as 0. The total score I of a DR-method is then

$$I = \sum_{i=1}^{n} \Big[\mathbb{1}_{\{\mathbb{P}(C'_i=c|C_i=c)>\mathbb{P}(C_i=c)\}} \cdot \log_2 \frac{\mathbb{P}(C'_i = c|C_i = c)}{\mathbb{P}(C_i = c)} + \mathbb{1}_{\{\mathbb{P}(C'_i=c|C_i=c)<\mathbb{P}(C_i=c)\}} \cdot \log_2 \frac{1 - \mathbb{P}(C_i = c)}{1 - \mathbb{P}(C'_i = c|C_i = c)} \Big].$$

Here, we used the equality $\log_2 x - \log_2 y = \log_2(x/y)$ for compactness. Usually, when comparing classifiers, I is reported relative to the expected information E needed to correctly classify each point:

$$E = -\sum_{i=1}^{n} \mathbb{P}(C'_i = c|C_i = c) \cdot \log_2(\mathbb{P}(C'_i = c|C_i = c)). \tag{3.17}$$

The Relative Information score I_r is then

$$I_r = \frac{I}{E} \cdot 100\%. \tag{3.18}$$

Note that I_r can become negative because I can also be negative. Inserting Eqs. (3.13)-(3.16) into Eq. (3.17) and Eq. (3.18) yields an expression in terms of the elements of the confusion matrix.

3.4 Data sets

In the previous sections we described the setup of our experiments, the DR-techniques, and how we measure their performance. The experiments use three real-world data sets which we describe below.

Radiant light energy measurements. The measurements in this data set are from sensor nodes deployed in several office buildings in New York City, as part of Columbia University's EnHANTs project. Each node has one sensor

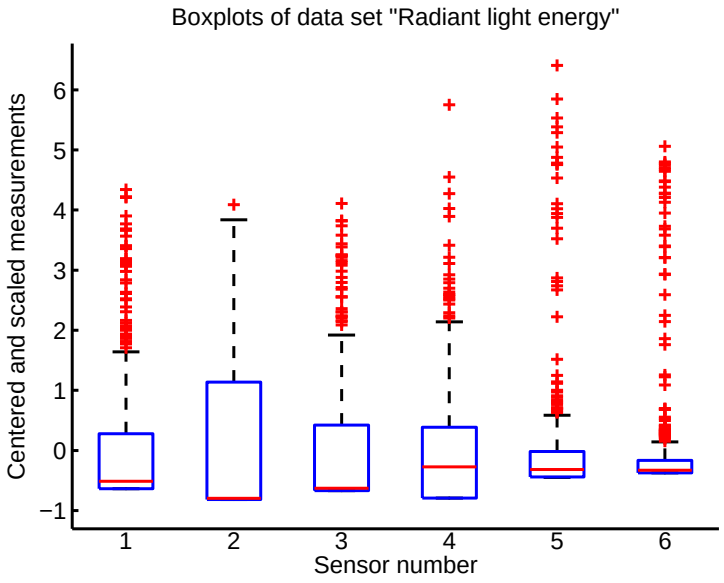


FIGURE 3.3: Boxplots of the sensors in the "Radiant light energy" data set.

measuring *irradiance* (radiant light energy), and this data set contains values measured during about one year. Figure 3.3 shows boxplots of each of the six sensors in this data set, with the measurements centered and scaled as discussed in Section 3.3. Each boxplot reflects the distribution of the 500 measurements by one sensor, and highlights possible outliers. Each sensor contains 40-60 possible outliers, except for the second sensor which has just one. The Peeling algorithm from Section 3.3.1 selects which of these points we use as outliers in our experiments. Note also that the median of each sensor's values (except sensor 4) is close to the .25 quantile, indicating that those distributions are skewed towards the smaller values.

More detailed information on the data set can be found in [57], or from the CRAWDAD website [58] where the data is available for downloading. For computational reasons, we do not use all the data for the experiments in this chapter, but select 500 random measurements from each of the six sensors.

Signal strength data. This data originates from a wireless sensor network deployed in a library building, where sensors measure radio frequency energy level (RSSI) on all 802.15.4 channels in the 2.4 GHz band [116, 117]. In essence, RSSI is an indication of the power level of a signal received by the antenna on the sensor node. The building has several collocated WIFI networks in normal

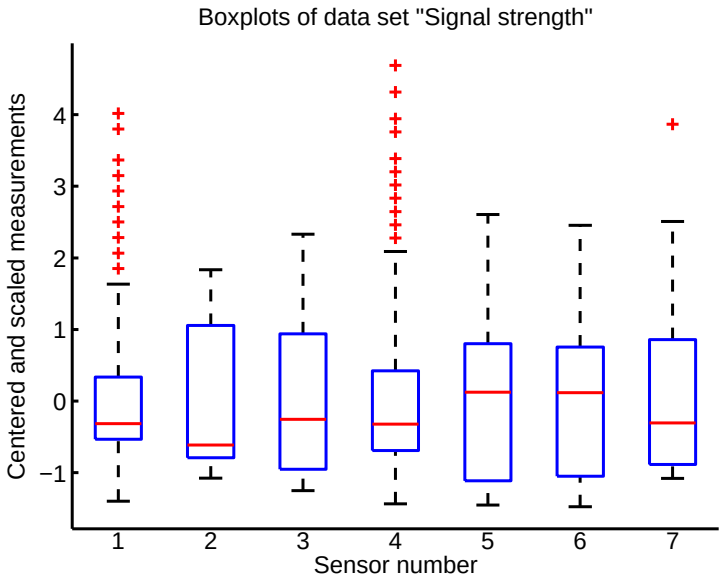


FIGURE 3.4: Boxplots of the seven sensors in the "Signal strength" data set.

operation that cause interference, so the sensor network is used to monitor the signal strengths on one location in this WIFI network. The sensor network consists of sixteen sensor nodes (each monitoring a single WIFI channel) of which we used only seven for computational reasons [25].

Again, we took 500 randomly selected measurements of each node to form this data set. The boxplots of the sensor values in this data set are in Figure 3.4. In contrast to the "Radiant light energy" data set, the measurements of the sensors in the "Signal strength" data set contain fewer possible outliers and are more evenly distributed. All possible outliers are positive values, corresponding to a strong incoming WIFI signal.

Decibel levels. This (proprietary) data set consists of five sensor nodes deployed in a kindergarten, one in each room of a single-story building, that are used to monitor the indoor climate. Among other parameters, the nodes measure decibel levels, and report these regularly to a central base station. We took 500 measurements from each decibel sensor on a day in May 2011 and included them in this data set. Figure 3.5 demonstrates that most sensors have fairly evenly distributed values, with several outliers on both sides of the median. However, kindergartens tend to be noisy rather than quiet, so most outliers are on the positive side of the median.

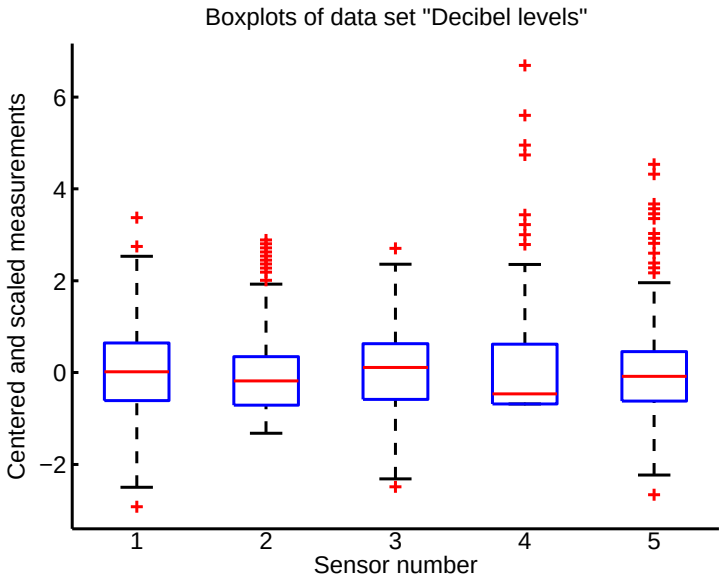


FIGURE 3.5: Boxplots of the five sensors in the "Decibel levels" data set.

3.5 Results and discussion

We apply the experimental setup of Section 3.3 to the DR-techniques of Section 3.2 and summarize the results in Table 3.1. This table contains the F1-score, Matthews Correlation, and Relative Information score for each combination of DR-technique and data set, where a high score implies that the technique preserves outliers well on that data set. Since MDS and t-SNE rely on a random initialization, we repeated the computation of scores 25 times, and reported the average and standard deviation of the results in Table 3.1. For the F1-scores, MDS achieves the best results, with values more than twice as large as those of PCA on the first and third data set. The scores of t-SNE are low, and suggest that it does not preserve outliers well. With the Matthews Correlation and Relative Information score we see similar results: MDS consistently attains high scores, PCA performs reasonably well on the first and third data set, and t-SNE has overall low scores.

We can visually inspect what happens to outliers after applying the three DR-methods. In Figures 3.6-3.8 we plot the low-dimensional version of the second data set "Signal Strength" (in circles), with outliers in the original high-dimensional data set marked by a triangle. Figure 3.6 demonstrates that

DR-technique	Light	Signal	Decibel
F1-score	*Avg(std)*	*Avg(std)*	*Avg(std)*
PCA	0.3333 (0.0000)	0.0000 (0.0000)	0.3529 (0.0000)
MDS	0.8316 (0.1076)	0.8705 (0.0828)	0.7212 (0.0739)
t-SNE	0.1067 (0.1587)	0.0073 (0.0364)	0.0352 (0.0718)
Matthews corr.			
PCA	0.3302 (0.0000)	-0.0149 (0.0000)	0.3477 (0.0000)
MDS	0.8439 (0.0907)	0.8767 (0.0754)	0.7374 (0.0617)
t-SNE	0.1389 (0.2146)	0.0033 (0.0472)	0.0497 (0.1170)
Rel. Inf. score			
PCA	0.7261 (0.0000)	-0.1295 (0.0000)	0.6398 (0.0000)
MDS	1.0197 (0.0465)	0.9118 (0.0299)	0.8409 (0.0238)
t-SNE	0.2583 (0.4720)	0.0020 (0.1364)	0.1228 (0.3021)

TABLE 3.1: F1-score ($\in [0,1]$), Matthews Correlation ($\in [-1,1]$), and Relative Information Score ($\in (-\infty,\infty)$) of each combination of DR-technique and data set. The reported values are the mean and standard deviation of 25 runs.

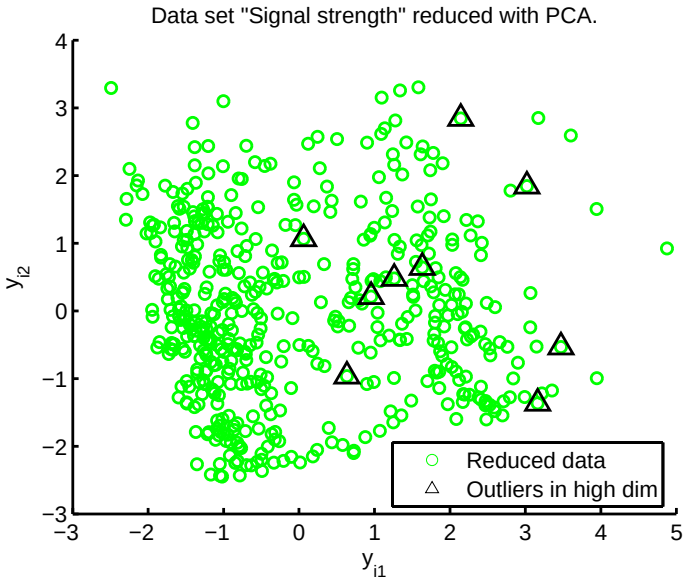


FIGURE 3.6: Data set "Signal strength" after dimensionality reduction with PCA (circles). The triangles mark the outliers that were found in the original high-dimensional data set.

most of the outliers are mapped to the interior of the reduced data set by PCA. In contrast, the low-dimensional data set in Figure 3.7 created by MDS has all high-dimensional outliers close to the boundary. Lastly, t-SNE also maps most outliers to the interior of the low-dimensional data set (shown in Figure 3.8), which illustrates its low scores.

By analyzing the objective of the three DR-techniques, we can explain the observed differences in performance. Firstly, PCA is a technique that focuses on preserving variance, so it only preserves outliers if they happen to be in a direction of high variance. Figure 3.1 from the introduction provides another illustration of what can happen to an outlier that is in a direction with low variance. The figure corresponds to reducing a two-dimensional data set to one dimension (the line) with PCA, and clearly demonstrates how the top-left outlier ends up in the center of the reduced data set.

MDS optimizes the squared stress optimization criterion in Eq. (3.4), which includes the term $||\mathbf{x}_i - \mathbf{x}_j||_M$. This term is the distance between two points $\mathbf{x}_i$ and $\mathbf{x}_j$, which is typically large when one of the points is an outlier. The criterion uses these distances to the power 4, so the outliers have a large effect on the squared stress criterion. Hence, minimizing these distances has a large positive effect on this criterion and thus MDS preserves outliers well.

The t-SNE technique optimizes the Kullback-Leibler divergence (3.8), which attaches high costs to nearby points in high-dimensional space (large p_{ij}) that are mapped to far away points in low-dimensional space (small q_{ij}). Hence, nearby points in high-dimensional space are kept nearby in low-dimensional space. This does not hold for points that are far away in high-dimensional space – outliers, which have low p_{ij} – as they are mapped to nearby points (with high q_{ij}) with low costs. So t-SNE tries to keep nearby points nearby and is therefore more suitable for preserving clusters than for preserving outliers.

From the analysis above we see that from the three selected methods, MDS achieves the highest scores and is best capable of preserving outliers. However, it is not necessarily the best DR-technique available, since many others exist in literature. In particular, the class of *supervised* DR-techniques (PCA, MDS, t-SNE are unsupervised) might provide methods with better performance than MDS. These techniques aim to reduce dimensionality while simultaneously trying to retain "sufficient information" for a classification task (which, in our case, would be retaining outliers). Hence, they could be applied to the scenario in this chapter, and possibly have good performance. Nevertheless, supervised DR-techniques are not included here, because we assume that the DR-techniques have no apriori knowledge about the outliers, and thus they are not suitable

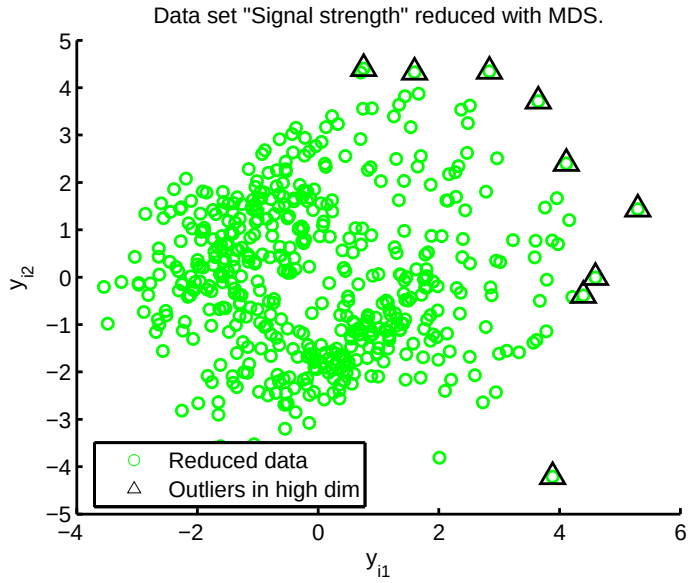


FIGURE 3.7: Data set "Signal strength" after dimensionality reduction with MDS (circles). The triangles mark the outliers that were found in the original high-dimensional data set.

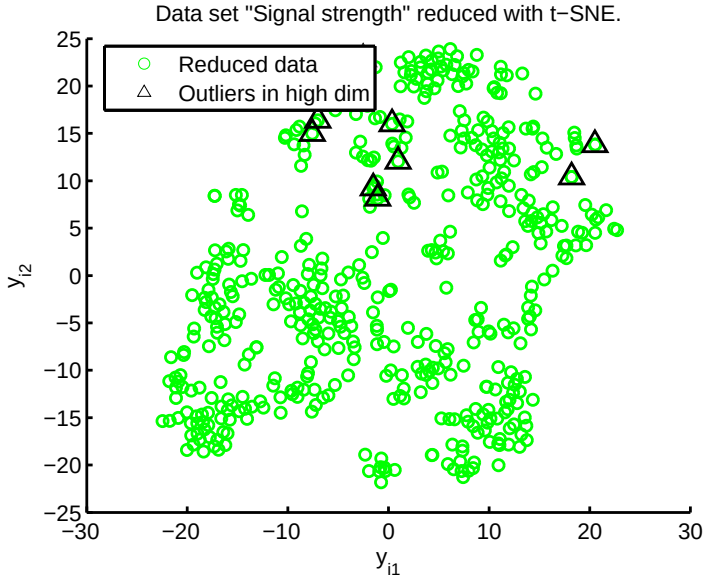


FIGURE 3.8: Data set "Signal strength" after dimensionality reduction with t-SNE (circles). The triangles mark the outliers that were found in the original high-dimensional data set.

for this chapter. Readers interested in supervised DR-techniques are referred to, e.g., [141].

The performance measures in this chapter are all based on the elements of the confusion matrix, which do not contain information about whether a point is a 'large' or 'small' outlier. Hence, with these scores we are not able to, e.g., find out which outlier has the large effect on a score. This 'binary' view of an outlier is, however, important for the scenario in the current chapter. Our motivation comes from applications where it is of critical importance to correctly identify an outlier after DR. If an outlier is no longer an outlier after DR, then it is useless for the application. Nevertheless, if this 'binary' approach can be relaxed from the point of view of the application, other scores might be more appropriate (see, e.g., [31]).

3.6 Conclusion

In this chapter we described three well-known DR-techniques (PCA, MDS, and t-SNE) and analyzed how well they are capable of preserving outliers. Based on three scores (F1-score, Matthews Correlation, and Relative Information score), and using three real-world data sets, we assessed the performance of each method on each data set. The resulting analysis demonstrates that, among the three described DR-methods, MDS is best at preserving outliers. It consistently achieves the highest scores, and performs significantly better than both PCA and t-SNE. In the discussion, we explain that this difference in performance is caused by the specific objectives of the techniques: PCA tries to preserve variance, MDS preserves large distances (i.e., outliers), and t-SNE preserves clusters. In general, we recommend that the dimensionality reduction technique is chosen with the intended application in mind. For outlier detection MDS is a good choice, PCA is designed for preserving variance, and for preserving clusters t-SNE is a good choice. Future research includes investigating specific types of dimensionality reduction (e.g., supervised DR-methods, real-time DR-methods), and how they are affected by outliers.

4

Throughput Modeling of the IEEE MAC for Sensor Networks

In this chapter we focus on the 'network' aspect of a 'sensor network'. The increasing number of sensors in a network result in large number of measurements that have to be transmitted to, e.g., a middleware component for further processing, thereby stressing the processing capabilities of the network. We consider a specific performance indicator of a network, namely the *saturation throughput*. This property reflects at what speed the network is able to process measurements by sensors when a large number of these measurements is offered. We provide a model for analyzing the saturation throughput of the IEEE 802.15.4 MAC protocol, which is the de-facto standard for wireless sensor networks, ensuring fair access to the channel. To this end, we introduce the concept of a *natural layer*, which reflects the time that a sensor node typically has to wait prior to sending a packet. The model is simple and provides insight how the throughput depends on the protocol parameters and the number of nodes in the network. Validation experiments with simulations demonstrate that the model is highly accurate for a wide range of parameter settings of the MAC protocol, and applicable to both large and small networks. As a by-product, we discuss fundamental differences in the protocol stack and corresponding throughput models of the popular 802.11 standard.

This chapter is based on the results presented in [9].

4.1 Introduction

The most widely used standard for sensor networks is the IEEE 802.15.4 protocol, which is aimed at providing low-cost, low-power communications for resource-limited devices. It is particularly suitable for sensor networks, since sensor nodes are typically battery powered and have few computational resources available. Part of this standard is the MAC protocol, which is responsible for governing access to the wireless channel. In particular, it describes the collision avoidance (CSMA-CA) mechanism employed by nodes to limit loss of packets due to collisions. In essence, this mechanism instructs nodes to wait a random amount of time before attempting a transmission. Without this mechanism, each node in the network would continuously attempt transmissions, causing massive loss of packets and large periods of inactivity on the network. However, the waiting time enforced by the CSMA-CA mechanism might decrease the throughput of the network significantly compared to the maximum specified in the standard.

In the literature, much work has been done on analyzing throughput of the IEEE 802.15.4 MAC protocol. This MAC protocol can be used in two different configurations: *slotted* and *unslotted*. The unslotted configuration is the simplest version, whereas the slotted protocol has richer features. Both configurations appear in the literature, and we review the state of the art below. The authors of [83] analyze both throughput and delay of the unslotted IEEE 802.15.4 MAC for a simple network containing a single node. They formulate an expression for the throughput and the delay in terms of the protocol parameters, and verify these with results from a real sensor network. [102] looks at the unslotted MAC in more detail and formulates a three dimensional Markov Chain for the CSMA-CA process. From this, expressions for link reliability, packet delay, and energy consumption are derived. The results are valid for both a network in star formation, as well as for a general multi-hop network. For large-scale networks an approximate model is constructed, in order to keep computations numerically tractable. Simulations are used to validate the model. [85] analyzes throughput for the unslotted MAC by combining a renewal process for the physical layer with a semi-Markov process for the MAC layer. The analysis results in equations that are solved via a fixed-point procedure, and the resulting throughput closely resembles values observed in a discrete event simulation.

For the slotted MAC, [126] is similar to [102]. The authors investigate throughput by constructing a two-dimensional Markov Chain, and derive an expression for the throughput. They then compare the results of the model with the outcome of simulations, and demonstrate that their model accurately captures the

throughput. In [89], Lee et al look at various performance metrics, including throughput and average service time for a transmission. Their method relies on viewing a cycle of a transmission and the subsequent waiting by a node as a renewal process. They derive a model that is solved via a fixed point iteration, and demonstrate its accuracy by comparing it to results from a discrete event simulation.

Although the papers mentioned above provide insight into the throughput behavior of sensor networks, the models involved are typically rather complex. Motivated by this, the goal of this chapter is to provide a simple yet accurate model for analyzing the throughput. To this end, we propose a new concept called the *natural layer* which reflects the time a sensor node typically has to wait as part of the CSMA-CA process prior to sending a packet (as detailed in Section 4.2.1). Using this concept, we develop a simple model for the throughput, and use simulation results to demonstrate that it is accurate for a wide range of realistic parameter settings. In our model, we focus on the unslotted version of the MAC protocol. The model provides insights into the differences between IEEE 802.15.4 and the popular 802.11 standard. In particular, we highlight the aspect of 'freezing' in the 802.11 protocol, and discuss how the absence of freezing in IEEE 802.15.4 influences the saturation throughput.

The organization of the chapter is as follows. In Section 4.2 we outline the IEEE 802.15.4 CSMA-CA protocol, list model assumptions and notation for our analysis. Then, in Section 4.3 we derive an expression for the throughput of sensor networks with a single backoff layer. Subsequently, in Section 4.4 we introduce the concept of a natural layer and use this to extend the model to a setting with multiple layers. In Section 4.5 we show simulation results to demonstrate that the model captures the throughput accurately for a wide range of parameter settings. Next, in Section 4.6 we discuss key differences between the IEEE 802.15.4 and 802.11 standards, and how these differences influence modeling of the throughput. Finally, Section 4.7 contains concluding remarks and ideas for future research.

4.2 Preliminaries

In this section we briefly outline the IEEE 802.15.4 CSMA-CA protocol, because a good understanding of this mechanism is essential when modeling throughput. Additionally, we list the model assumptions and provide some preliminary remarks.

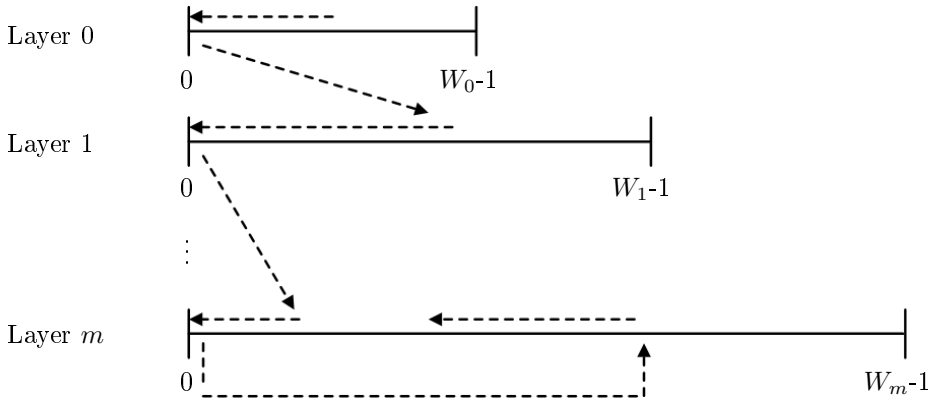


FIGURE 4.1: The CSMA-CA mechanism applied by each node in the network.

4.2.1 The IEEE 802.15.4 CSMA-CA protocol

The CSMA-CA mechanism states that, in order to avoid collisions, a node should wait a random amount of time (known as the *backoff time*) prior to sending a packet. This waiting time affects throughput, and a thorough understanding of the CSMA-CA mechanism is key to modeling throughput. The backoff time is drawn uniformly from the interval $[0, W_0 - 1]$, where W_0 is the initial *backoff window* (controlled via parameter *macMinBE*). The resulting backoff time is discrete, and corresponds to the number of time slots that the node has to wait. The length (in seconds) of a time slot is defined in IEEE 802.15.4. After the required backoff time, the node assesses if the channel is idle and if it is, the node sends the packet. If, however, the channel is busy, the window W_0 is doubled and the backoff process starts again. This procedure is repeated until the packet is sent.

Initially, the backoff window W_0 is set to $2^{macMinBE}$ and it is repeatedly doubled during the CSMA-CA process. However, overly long backoff times cause unnecessary delays, so the CSMA-CA mechanism defines a maximum backoff exponent (*macMaxBE*). Once the backoff window reaches $2^{macMaxBE}$, the doubling is disabled.

Figure 4.1 illustrates the CSMA-CA mechanism as described above. The node starts in layer 0 and draws a random backoff time from the interval $[0, W_0 - 1]$. Then it waits until this backoff time has passed, and does a channel assessment. If the channel is busy, the process moves to layer 1. The window in layer 1 is twice as large as that of layer 0 ($W_1 = 2 \cdot W_0$) because of the doubling of the window. The node now draws a backoff time from $[0, W_1 - 1]$, and

again waits until this time has passed. The window is repeatedly doubled until the backoff exponent reaches *macMaxBE* (layer m), at which point the doubling is disabled. The CSMA-CA process then continues until the packet is sent successfully.

4.2.2 Assumptions

Before starting our throughput analysis we mention several assumptions we make in this chapter.

- Sensor nodes are structured as a star network, and all nodes send packets to the sink node.
- The network is saturated, meaning that nodes always have a packet ready for transmission. Consequently, there are no periods of inactivity on the channel caused by a lack of packets.
- The network uses non-beacon mode, and the unslotted version of the CSMA-CA mechanism.
- Acknowledgements are disabled.
- If two nodes finish a backoff cycle simultaneously, only one of the packets is transmitted. The other packet moves to the next backoff layer. Hence, there are no packet collisions. In Section 4.6 we revisit this topic.
- Packets go through the CSMA-CA mechanism until they are sent.

4.2.3 Preliminary remarks

Notation. The IEEE 802.15.4 MAC variables *macMinBE* and *macMaxBE* are cumbersome in a mathematical analysis, so we use a different notation in the remainder of this chapter. By W_0 we denote the backoff window for layer 0, i.e., $W_0 = 2^{macMinBE}$. We use m instead of "*macMaxBE*-*macMinBE*" to indicate how often layer 0 is doubled in size. Finally, T is the number of time slots (see below) required to send one packet.

Continuous backoff time. For this chapter, we assume that the random backoff times in the CSMA-CA mechanism are drawn from a *continuous* uniform distribution, even though IEEE 802.15.4 specifies a *discrete* uniform distribution. This is done purely for notational convenience, and our method works for discrete uniform distributions as well.

Channel speed. We set the channel speed (and thus the maximum throughput) to 250,000 b/s. The IEEE 802.15.4 standard specifies several options for the channel speed, depending on configuration and geographical location. Our choice for the channel speed is not essential to the model in this chapter; it works for other channel speeds as well. To emphasize this, we always normalize the throughput to the interval $[0, 1]$ when reporting on it.

Unit of time. It seems natural to report on time in units of seconds, but this has several drawbacks. First, the time scales involved are small (in the order of fractions of milliseconds), and are thus somewhat laborious to work with. Second, the times depend on the speed of the channel, and this can vary per configuration and per geographical region. Even though we choose a certain channel speed in this chapter, our analysis works for other choices as well. To preserve this neutrality, we use the time slots from the CSMA-CA process as the unit of time throughout this chapter. These time slots are configuration- and region-neutral, and can easily be converted to seconds if needed (this is described in the IEEE 802.15.4 standard). An additional benefit of using time slots is that we can quickly compare, e.g., the time needed to transmit a packet to the waiting times described by the CSMA-CA process. Finally, note that a non-integer number of time slots is also meaningful when using them as unit of time – for instance, a single packet transmission takes 12.7 time slots.

4.3 Single-layer analysis

We start our throughput analysis by looking at a simplified version of the CSMA-CA mechanism. In this section we assume that it uses just one layer, layer 0. This simplified scenario forms an introduction to the complete throughput analysis, later in this chapter. Figure 4.2 shows n nodes going through the CSMA-CA process of transmitting packets (marked by T) and backing off (denoted by $u_{1,1}, \ldots, u_{n,2}$). The interval lengths $u_{1,1}, \ldots, u_{n,2}$ are backoff times drawn from the uniform distribution on interval $[0, W_0 - 1]$ (all nodes are at backoff layer 0 in our simplified scenario). Node 1 is the first to send a packet, and during the transmission at node 1, the other nodes are going through several backoff cycles. In particular, node 2 starts three backoff cycles (of length $u_{2,1}, u_{2,2}, u_{2,3}$, respectively), and node n starts two cycles (of length $u_{n,1}$ and $u_{n,2}$). At the end of the transmission at node 1, this node also starts a backoff cycle (of length $u_{1,1}$). The first backoff cycle to end after the transmission at node 1 is the one of length $u_{2,3}$ at node 2, so the next transmission occurs at node 2. This process continues over time.

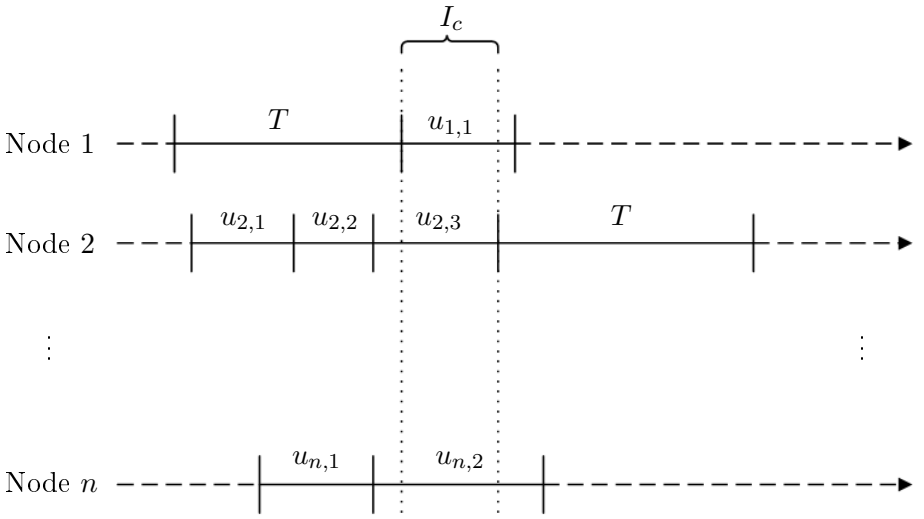


FIGURE 4.2: Events in the CSMA-CA process in between two transmissions (marked by T). The interval lengths $u_{1,1}, \ldots, u_{n,2}$ are backoff times drawn from the uniform distribution on interval $[0, W_0 - 1]$.

For determining throughput, we analyze the time that the channel is idle in between two transmissions. In Figure 4.2, this idle time (denoted by I_c) is the time between the end of the transmission at node 1, and the start of the next transmission at node 2. Note that at the end of the packet transmission at node 1, the other nodes have already partly gone through a backoff cycle. Node 1, in contrast, starts a new backoff cycle. Our throughput analysis takes these two aspects into account: we determine the distribution of the channel idle time using the distribution of the length of the backoff cycle at node 1, and using the distribution of the residual of the backoff cycles at the other nodes.

To start the analysis we introduce some notation. $U_0^{(k)}$ is the random variable representing the backoff time at layer 0 for node k ($1 \leq k \leq n$). It is uniformly distributed (continuously, by assumption) on the interval $[0, W_0 - 1]$. The corresponding probability density function (pdf) and cumulative distribution function (cdf) are denoted by $f_{U_0^{(k)}}(t)$ and $F_{U_0^{(k)}}(t)$, respectively. Next we formulate the distribution of the residual of $U_0^{(k)}$. To be precise, suppose a node starts a backoff cycle of length t and the node that is currently sending a packet finishes at time $s \in [0, t]$. We are interested in the distribution of the residual backoff time $t - s$. We denote this residual backoff time by $\bar{U}_0^{(k)}$, with

pdf $f_{\bar{U}_0^{(k)}}(t)$ and cdf $F_{\bar{U}_0^{(k)}}(t)$. The density of $\bar{U}_0^{(k)}$ is given by

$$f_{\bar{U}_0^{(k)}}(t) = \frac{1 - F_{U_0^{(k)}}(t)}{\mathbb{E}U_0^{(k)}}, \quad t > 0, \quad 1 \leq k \leq n, \tag{4.1}$$

which is the well-known distribution of the residual backoff time [13]. For the throughput analysis we are interested in the idle time of the channel, i.e., the time in between the end of a transmission, and the start of the next one. If we assume, without loss of generality, that node 1 is the one finishing a transmission, then the idle time involves random variable $U_0^{(1)}$ (for node 1), and $\bar{U}_0^{(2)}, \ldots, \bar{U}_0^{(n)}$ for the remaining nodes. We are looking for the first backoff cycle to finish, i.e., the expectation of the minimum of these n random variables. The idle time of the channel, I_c, is then given by $I_c = \min\{U_0^{(1)}, \bar{U}_0^{(2)}, \ldots, \bar{U}_0^{(n)}\}$. We determine $F_{I_c}(t)$, the cdf of I_c, for $t > 0$ via:

$$\begin{aligned}
F_{I_c}(t) &= \mathbb{P}(I_c \leq t) \\
&= \mathbb{P}(\min\{U_0^{(1)}, \bar{U}_0^{(2)}, \ldots, \bar{U}_0^{(n)}\} \leq t) \\
&= 1 - \mathbb{P}(\min\{U_0^{(1)}, \bar{U}_0^{(2)}, \ldots, \bar{U}_0^{(n)}\} \geq t) \\
&= 1 - \mathbb{P}(\min\{U_0^{(1)}, \bar{U}_0^{(1)}, \ldots, \bar{U}_0^{(1)}\} \geq t) \\
&= 1 - \mathbb{P}(U_0^{(1)} \geq t) \cdot \big(\mathbb{P}(\bar{U}_0^{(1)} \geq t)\big)^{n-1} \\
&= 1 - (1 - F_{U_0^{(1)}}(t)) \cdot (1 - F_{\bar{U}_0^{(1)}}(t))^{n-1} \\
&= 1 - \mathbb{E}U_0^{(1)} \cdot f_{\bar{U}_0^{(1)}}(t) \cdot (1 - F_{\bar{U}_0^{(1)}}(t))^{n-1}.
\end{aligned} \tag{4.2}$$

In the fifth equality in Eq. (4.2) we used independence of the random variables $U_0^{(1)}, \bar{U}_0^{(2)}, \ldots, \bar{U}_0^{(n)}$, and in the last equality we substituted Eq. (4.1). The expectation of I_c can now be obtained by integrating the tail probabilities:

$$\begin{aligned}
\mathbb{E}I_c &= \int_0^{W_0-1} \mathbb{P}(I_c \geq t)dt \\
&= \int_0^{W_0-1} \big(1 - F_{I_c}(t)\big)dt \\
&= \int_0^{W_0-1} \mathbb{E}U_0^{(1)} f_{\bar{U}_0^{(1)}}(t) \cdot (1 - F_{\bar{U}_0^{(1)}}(t))^{n-1} dt \\
&= \frac{\mathbb{E}U_0^{(1)}}{n} \\
&= \frac{W_0 - 1}{2n},
\end{aligned} \tag{4.3}$$

where we used in that last equality that $U_0^{(1)}$ is uniform on $[0, W_0-1]$. Conform our expectation, with $n = 1$ the waiting time is half the initial backoff window W_0, and as $n \to \infty$ the waiting time tends to 0. The throughput S_c is now computed using

$$S_c = \frac{T}{T + \mathbb{E}I_c} = \frac{T}{T + \frac{W_0-1}{2n}}, \tag{4.4}$$

with T the number of time slots needed to transmit a single packet. Numerical experiments in Section 4.5 demonstrate that this expression does indeed capture the throughput accurately.

4.4 Multi-layer analysis

When the assumption of a single layer is dropped, the situation becomes considerably more complex. At the end of the packet transmission we now no longer know the distribution of the remaining $n-1$ nodes. For instance, after the packet transmission at node 1 in Figure 4.2, node 2 has been through two backoff cycles and is busy with at least its third backoff cycle. It might even be more than that, since the figure does not show what happened at node 2 prior to the cycle of length $u_{2,1}$. Potentially, a throughput analysis of the multi-layer scenario involves a large and complex model including the behavior of individual nodes. Clearly, such models are intractable for larger networks with many nodes. In this section we provide a simple model for the throughput that allows us to overcome this issue. Before continuing, we recall that in a multi-layer scenario, the backoff window depends on the layer. To be precise, in layer x the window W_x is

$$W_x = W_0 \cdot 2^{\min(x,m)}. \tag{4.5}$$

So each time a node moves to the next backoff layer, the window is doubled, until it reaches layer m. At layer 0, the window is W_0, and at layer $x \geq m$ the window is $W_0 2^m$. See also the description of the CSMA-CA protocol in Section 4.2.1. For now, suppose that at the end of a packet transmission, the $n-1$ other nodes are all at the same layer, and denote this layer by x. Node 1 (which just finished the transmission) is at layer 0. Following the notation of the previous section, we denote the backoff time at layer x for node k ($1 \leq k \leq n$) by $U_x^{(k)}$, and the corresponding remainder by $\bar{U}_x^{(k)}$. Then, we have $I_c(x) = \min\{U_0^{(1)}, \bar{U}_x^{(2)}, \ldots, \bar{U}_x^{(n)}\}$. Note that we changed notation from I_c to $I_c(x)$, reflecting the dependency on layer x. Repeating the steps of

the previous section gives $F_{I_c(x)}(t)$ for $t > 0$:

$$\begin{aligned} F_{I_c(x)}(t) &= \mathbb{P}(I_c(x) \leq t) \\ &= \mathbb{P}(\min\{U_0^{(1)}, \bar{U}_x^{(2)}, \ldots, \bar{U}_x^{(n)}\} \leq t) \\ &= 1 - \mathbb{P}(\min\{U_0^{(1)}, \bar{U}_x^{(2)}, \ldots, \bar{U}_x^{(n)}\} \geq t) \\ &= 1 - \mathbb{P}(\min\{U_0^{(1)}, \bar{U}_x^{(1)}, \ldots, \bar{U}_x^{(1)}\} \geq t) \\ &= 1 - \mathbb{P}(U_0^{(1)} \geq t) \cdot \left(\mathbb{P}(\bar{U}_x^{(1)} \geq t)\right)^{n-1} \\ &= 1 - (1 - F_{U_0^{(1)}}(t)) \cdot (1 - F_{\bar{U}_x^{(1)}}(t))^{n-1} \\ &= 1 - \mathbb{E}U_0^{(1)} \cdot f_{\bar{U}_0^{(1)}}(t) \cdot (1 - F_{\bar{U}_x^{(1)}}(t))^{n-1}. \end{aligned} \tag{4.6}$$

We can also compute $\mathbb{E}I_c(x)$ as before by integrating the tail probabilities via

$$\mathbb{E}I_c(x) = \int_0^{W_0-1} \mathbb{P}(I_c(x) \geq t)dt. \tag{4.7}$$

Here, we used that $U_0^{(1)}$ is uniform on $[0, W_0 - 1]$ to establish the interval over which to integrate. The resulting expression for $\mathbb{E}I_c(x)$ requires several pages to display, so we omit it here for compactness. The expression for throughput in Eq. (4.4) still holds, but we repeat it here with adapted notation to emphasize the dependence on x:

$$S_c(x) = \frac{T}{T + \mathbb{E}I_c(x)}. \tag{4.8}$$

We now focus our attention on the throughput analysis of a single node. Without loss of generality, we assume that this is node 1. Prior to a packet transmission, node 1 spent some time waiting as part of the CSMA-CA process. By assumption, we know that it is currently at layer x and thus we also know how much time node 1 spent waiting: the sum of the expected backoff time at layers $0, \ldots, x$. However, for reasons that become apparent later, we need a sensible interpretation of a layer number x that is non-integer. To this end, suppose that $x = \lfloor x \rfloor + \alpha$, with $\alpha \in [0, 1)$ and $\lfloor x \rfloor$ the largest integer smaller than x. When a node is at a decimal layer x, we interpret this as it having to wait at all integer layers $0, \ldots, \lfloor x \rfloor$, plus a fraction α at the layer with backoff window $W_{\lfloor x \rfloor + \alpha}$. This interpretation is consistent with the integer view of layers when $\alpha = 0$ and when α tends to 1.

With this interpretation, we denote the waiting time on a node by $I_N(x)$ and calculate its expectation from

$$\mathbb{E}I_N(x) = \sum_{j=0}^{\lfloor x \rfloor} \mathbb{E}U_j^{(1)} + \alpha \mathbb{E}U_x^{(1)}. \tag{4.9}$$

We can expand the sum in Eq. (4.9) further, taking care that the doubling is stopped after layer m and that we do not know whether x is larger or smaller than m. With $U_x^{(1)}$ uniformly distributed on $[0, W_x - 1]$, some careful calculations yield

$$\begin{aligned} \mathbb{E}I_N(x) &= -\frac{\lfloor x \rfloor + 1}{2} + W_0 \frac{2^{\min(\lfloor x \rfloor, m)+1} - 1}{2} \\ &+ \frac{W_0 2^m}{2}(\lfloor x \rfloor - m)^+ + \alpha W_0 \frac{2^{\min(x,m)} - 1}{2}, \end{aligned} \tag{4.10}$$

where $(x)^+ = \max(0, x)$. Similar to Eq. (4.4), we can find the throughput of one particular node using

$$S_N(x) = \frac{T}{T + \mathbb{E}I_N(x)}. \tag{4.11}$$

We now have an expression for the throughput on the channel from $S_c(x)$ in Eq. (4.8), and for the throughput provided by each node ($S_N(x)$ in Eq. (4.11)). In a fair star network, all nodes are identical and each contributes an equal share to the throughput on the channel. Therefore, the following consistency relation should hold:

$$S_c(x) = n \cdot S_N(x). \tag{4.12}$$

Analyzing the saturation throughput is now done by calculating a value x such that Eq. (4.12) holds.

In Figure 4.3, the throughput expressions for $n \cdot S_N(x)$ and $S_c(x)$ are plotted for a network with $n = 10$ nodes. We see that for $x = 0$, $n \cdot S_N(x) > S_c(x)$ and that $n \cdot S_N(x)$ decreases to 0 as x increases. $S_c(x)$, on the other hand, becomes constant as x increases, and the two lines intersect at the dotted line. We are looking for the value of x for which this intersection occurs (denoted by x^*). The following lemma shows that x^* exists and is unique.

LEMMA 4.4.1. *The consistency relation in Eq. (4.12) has a unique solution x^*.*

PROOF. We begin the proof by inspecting Eq. (4.12) with $n = 1$, for which it reduces to $\mathbb{E}I_N(x) = \mathbb{E}I_c(x)$. For $\mathbb{E}I_c(x)$, Eq. (4.7) is the same as Eq. (4.3) so that $\mathbb{E}I_c(x) = (W_0 - 1)/2$. The same expression results from Eq. (4.10) if we calculate $\mathbb{E}I_N(0)$, and thus for $n = 1$ the natural layer is $x^* = 0$. This corresponds to intuition, since with a network containing 1 node, the channel is always free at the end of the backoff time at layer 0, and there is no need to go to higher layers.

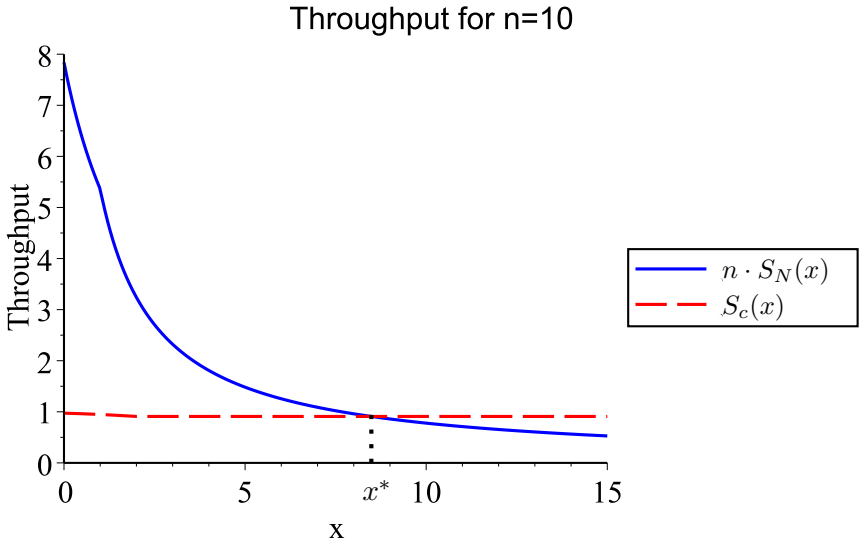


FIGURE 4.3: Throughput $n \cdot S_N(x)$ and $S_c(x)$ for a network with $n = 10$ nodes. $S_N(x)$ decreases as x increases, whereas $S_c(x)$ becomes constant. The x value at which the two lines intersect is the natural layer x^*.

To show uniqueness for the case with $n > 1$ we inspect the behavior at $x = 0$ and as $x \to \infty$. At $x = 0$ we have $\mathbb{E}I_c(0) = \frac{W_0-1}{2n}$ (again from Eq. (4.3)), and thus

$$S_c(0) = \frac{T}{T + \frac{W_0-1}{2n}} = \frac{n \cdot T}{nT + \frac{W_0-1}{2}}.$$

From Eq. (4.10) we get $\mathbb{E}I_N(0) = \frac{W_0-1}{2}$ and thus

$$n \cdot S_N(0) = \frac{n \cdot T}{T + \frac{W_0-1}{2}}.$$

So at $x = 0$ we have $n \cdot S_N(0) > S_c(0)$ (since $n > 1$).

As $x \to \infty$, $\mathbb{E}I_N(x)$ tends to infinity linearly, and thus $S_N(x)$ tends to 0. However, $\mathbb{E}I_c(x)$ becomes constant as $x \to \infty$, because the doubling of W_x is stopped when $x > m$. Hence, $S_c(x)$ also tends to a constant and as $x \to \infty$ we have $n \cdot S_N(x) < S_c(x)$. Consequently, by the Intermediate Value Theorem [123], somewhere in the interval $(0, \infty)$, there is a unique $x = x^*$ where the monotonously decreasing $n \cdot S_N(x)$ crosses the constant $S_c(x)$, so that we have $S_c(x^*) = n \cdot S_N(x^*)$.

□

Definition We call the unique solution x^* to Eq. (4.12) the *natural layer*. Based on Eq. (4.10), the natural layer is interpreted as the expected amount of time that a node typically has to wait as part of the CSMA-CA process, prior to sending a packet.

Observe that there is no guarantee that the natural layer x^* is an integer, which is why we extended the interpretation of a layer to non-integer values. In the next section we demonstrate that the throughput $S_c(x^*)$ closely resembles the results of simulations.

4.5 Experiments

We validate the model described in the previous section by comparing it to the results obtained from a discrete event simulation of the CSMA-CA process. Finding the throughput using our model is done by numerically finding the natural layer x^* for which Eq. (4.12) holds. Once x^* is found, we use Eq. (4.8) to calculate $S_c(x^*)$.

In Figure 4.4 we compare the throughput S_c obtained from our model (lines), to the results of the discrete event simulations (markers). The figure shows the throughput for varying number of nodes n, and several IEEE 802.15.4 parameter settings (for easy notation we report W_0 and W_m instead of the corresponding parameter values for *macMinBE* and *macMaxBE*). The values from the analysis closely match those of the simulations, demonstrating that our analysis accurately captures the throughput. Also, as n increases the throughput tends to 1 for all parameter settings. This is as we expected, since we assumed in Section 4.2.2 that there are no collisions between packets that simultaneously finish a backoff cycle. We revisit the topic of collisions in the next section.

Note that with $n = 1$ the natural layer is always $x^* = 0$ and the multi-layer analysis in Section 4.4 should match the result of the single-layer analysis in Section 4.3. Substituting $n = 1$ in Eq. (4.4) yields the values in Table 4.1, which nicely match the left-most markers in Figure 4.4. Next, we inspect the natural layer numbers corresponding to the graphs in Figure 4.4. These are plotted in Figure 4.5, where for small n the lines show a slight curvature, and as n increases they suggest a linear increase in the natural layer. These effects are due to the MAC protocol stopping the doubling of the backoff window after layer m. We expect that with a deeper analysis we are able to explain the effects in detail.

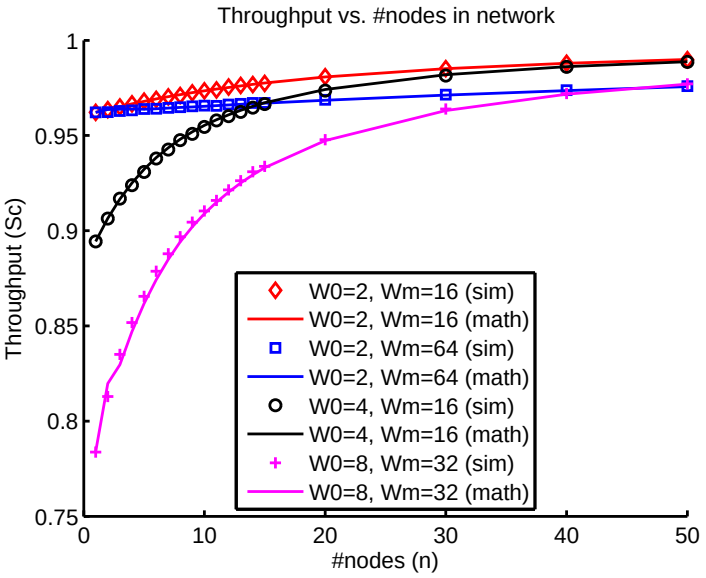


FIGURE 4.4: Throughput S_c as computed via the natural layer (solid line), and as obtained from simulations (markers), for varying number of nodes in the network (n).

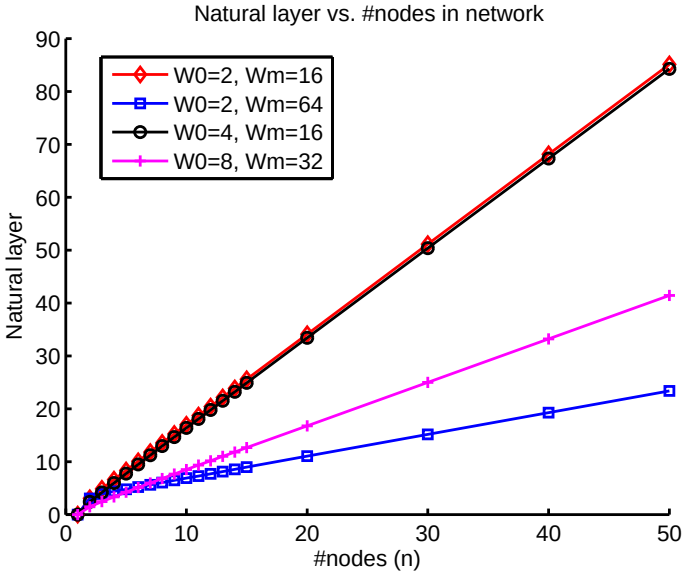


FIGURE 4.5: The natural layer x^* as computed in our throughput analysis for varying number of nodes in the network (n).

W_0	W_m	S_c
2	16	0.96
2	64	0.96
4	16	0.89
8	32	0.78

Table 4.1: Throughput according to the single-layer analysis from Eq. (4.4) for the parameter values used in Figure 4.4.

Figures. 4.4-4.5 demonstrate that despite its simplicity the model leads to an accurate prediction of the throughput for a wide range of protocol parameter settings. In the next section we make several remarks relevant to the throughput model discussed in this chapter.

4.6 Discussion

Comparison to 802.11. Many papers investigating saturation throughput are based on the paper by Bianchi [29], who formulates a model for the throughput of a WLAN network as specified in the IEEE 802.11 standard. The MAC protocols of IEEE 802.11 and IEEE 802.15.4 are highly similar, except for a property called *freezing*. In 802.11, a node that is backing off does a channel assessment at the end of each time slot, to see if the channel is busy. If the channel is busy, the backoff process is paused until the channel is free again. So during a transmission, all non-sending nodes are idle and not backing off. This 'freezing' feature, which is absent in IEEE 802.15.4, has significant impact on the throughput performance of 802.11.

Specifically, consider the three event types on the channel identified by Bianchi: a successful transmission, a collision between two or more packages, and a backoff event. The probability of these events is easy to derive from the two-dimensional discrete-time Markov chain (defined in [29]) that describes the evolution of the backoff state (i, k), where i is the retransmission counter and k is the backoff counter. For the IEEE 802.15.4 protocol the absence of freezing implies that the Markov chain has to be extended to include the duration of transmissions (as done in, e.g., [126]). As an alternative, in the present chapter we propose a different approach by introducing the concept of a natural layer, allowing us to consider a much simpler, single-layered model for the throughput.

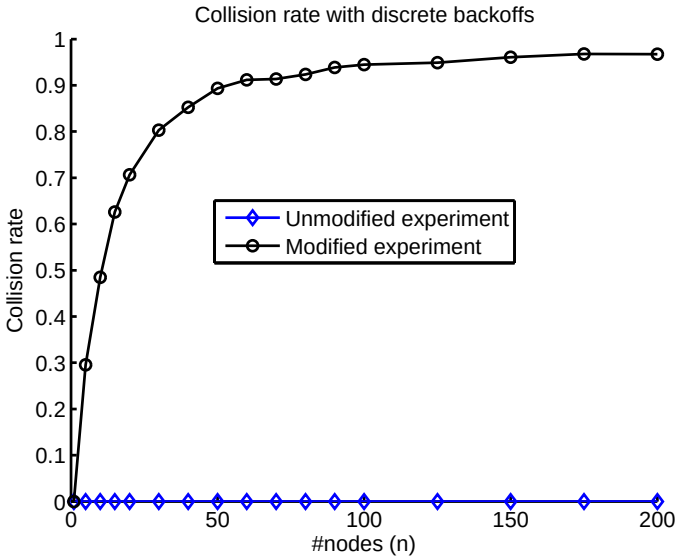


FIGURE 4.6: Collision rate when a discrete backoff time distribution is used in simulations. The experiment shows a zero collision rate (diamonds) because (1) a random waiting time is used prior to sending the first packet, and (2) a packet transmission is equivalent to a non-integer number of backoff steps. When the first is disabled, and the second modified to an integer, the collision rate is high and steadily approaches 1 as the number of nodes increases (circles).

Collisions. A collision between packets occurs when two or more nodes finish backing off at the same time slot, see the channel idle, and consequently transmit a packet simultaneously. In the scenario with continuously distributed backoff times (as we assume in this chapter), it is highly unlikely that two or more nodes finish backing off at the same time and cause a collision. At first glance, this event seems more likely in the scenario with a discrete distribution for the backoff times (as used in IEEE 802.15.4). If two nodes draw the same (discrete) backoff time and start the backoff process at the same time, then they potentially cause a collision.

There are, however, two factors that make it unlikely that the two backoff processes start simultaneously. First, our discrete event simulation waits a random amount of time before processing the first packet. This amount of time is drawn from a continuous uniform distribution, thereby preventing the backoff process at the nodes to start simultaneously. In practice, such a precaution is advised as well. Second, a packet transmission is equivalent to a non-integer number of backoff steps.

To verify this observation about collisions, we change the distribution of the backoff time to a discrete uniform distribution, run simulations again, and record the collision rate. The result is plotted in Figure 4.6 (blue diamonds), and demonstrates that the collision rate is equal to 0, even for a large number of nodes. Next, we disable the random waiting time that is used before processing the first packet, and modify the packet size such that a transmission takes an integer number of backoff steps. Figure 4.6 shows a collision rate that approaches 1 as the number of nodes n in the network increases (black circles). Hence, with the disabled waiting time and the modified packet size, most transmissions cause a collisions as expected. This experiment demonstrates that the inclusion of a random waiting time prior to the first packet transmission and of the non-integer number of backoff steps needed for a packet transmission, effectively prevent collisions.

Avoiding long waiting times. Figure 4.4 demonstrates that our model for the throughput accurately captures the throughput recorded in a discrete event simulation. However, a close look at the line corresponding to parameter values $W_0 = 8, W_m = 32$ (the IEEE 802.15.4 defaults) suggests a slight irregularity for our model at $n = 2$. This irregularity is exaggerated in Figure 4.7, where we decrease the packet size from 1,250 to 250 bits and fix $W_0 = 8, W_m = 32$. For small packet sizes, our model only captures throughput well for large n.

The irregularity is due to a small packet size as compared to the backoff times. For example, suppose node 0 starts a packet transmission and the other $n-1$ nodes are backing off. Node 0 then transmits the packet, and draws a new backoff time from interval $[0, W_0 - 1]$. If the sum of this transmission time and backoff time is smaller than the residual backoff times at the other $n-1$ nodes, node 0 also transmits the next packet. Hence, in a scenario where the packet size is small and the residual backoff times are large, it is likely that several consecutive transmissions occur at node 0. We observed the tendency for consecutive transmissions in the discrete event simulation as well. The irregularity vanishes for increasing n, since then the minimum of the residual backoff times at the other $n-1$ nodes decreases.

Our model assumes, in Eq. (4.6), that the random variables for the residual backoff time $\bar{U}_{x^*}^{(k)}$ of the $n-1$ waiting nodes are independent. In the situation described above, this assumption fails and our model no longer captures the throughput well. This is, however, not a severe restriction on our model: if a network operator expects mainly small packets, he has the option to choose appropriately small values for the window sizes, thereby avoiding situations with long waiting times. Our model can be used to find values for the protocol

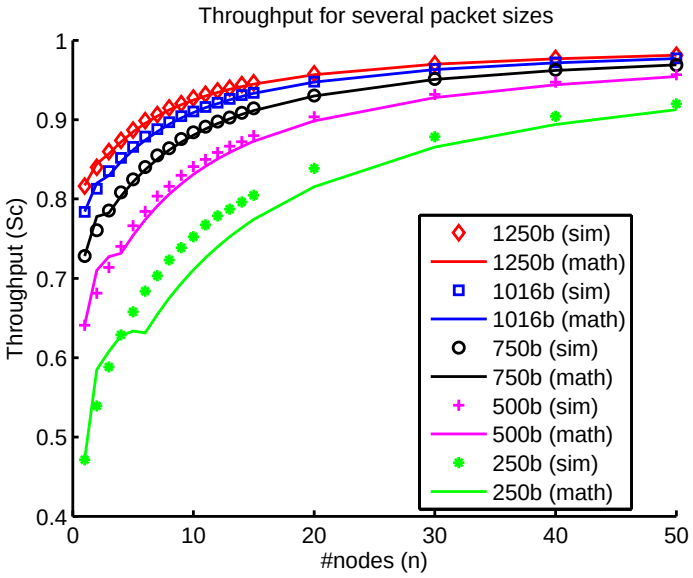


FIGURE 4.7: Throughput S_c via the natural layer (solid line), and simulations (markers), for decreasing packet size.

parameters such that waiting times are acceptable.

Near insensitivity to the backoff time distribution. Our throughput model is also valid for non-uniform backoff time distribution. Section 4.4 is written for general $U_x^{(k)}$, and only requires a change to the lower and upper bound of the integral in Eq. (4.7) if the distribution has a domain different from $[0, W_0]$. On the node level, Eqs. (4.9) and (4.11) remain valid, as does the consistency relation Eq. (4.12).

4.7 Conclusion

In this chapter we presented a simple yet powerful method for analyzing the throughput of a network of sensor nodes running the IEEE 802.15.4 MAC protocol. We introduced the concept of a *natural layer* which allowed us to analyze the waiting time involved in the MAC protocol. Then, we formulated a model for the throughput on the channel, and a model for the contribution to the throughput of a single node. Combining these two resulted in an equation from which we numerically computed the natural layer, which in turn gave

the throughput. The model was validated with experiments from a discrete event simulation, and demonstrated that our model accurately captures the throughput from the simulations.

Future work includes adding more features of the MAC protocol, particularly acknowledgements and a maximum number of layers in the CSMA-CA process. Central to this research will be analyzing how much extra idle time these aspects cause in the wireless channel, and how they influence the natural layer.

Part II

Markov Decision Processes

5

On the Control of a Queueing System with Aging State Information

In previous chapters we considered various aspects of sensor networks and the data produced by these networks. The following chapters deal with *Markov Decision Processes* (MDPs), a popular modelling framework for scenarios involving sequential decision making under uncertainty. For the current chapter we investigate control of a queueing system in which a component of the state space is subject to aging. The controller can choose to forward incoming queries to the system (where it needs time for processing), or respond with a previously generated response (incurring a penalty for not providing a fresh value). Hence, the controller faces a trade-off between data freshness and response times.

A similar trade-off occurs in the context of wireless sensor networks. Consider, e.g., a scenario where sensor nodes periodically report measurements to a centralized middleware component, which then saves these reports into, e.g., a database. When an application needs a measurement from a sensor, it sends a query to the middleware component. The component (having the role of controller) can then decide to either fetch a new measurement from the network, or to return a previously generated measurement from the database. Obtaining fresh measurements from the wireless sensor network is time-consuming due to the wireless transmissions across the network. On the other hand, the latest value in the database might be too old for practical purposes. The middleware component thus faces a trade-off that is similar to the one discussed in this

chapter. In practice, middleware components typically use a simple threshold policy, where the network is used when the age of the last reported value exceeds some specified threshold.

For the current chapter we consider a scenario involving a more general queueing system, as an illustration of how a simple threshold policy can be improved by taking the load of the network into account. We model the system as a Markov Decision Process that turns out to be complex, then simplify it, and construct a control policy. This policy has near-optimal performance and achieves lower costs than both a threshold policy and a myopic policy.

This chapter is based on the results presented in [6] and [2].

5.1 Introduction

We illustrate the system in Figure 5.1, where a controller *Ctrl* handles incoming queries that require a response. The controller uses a policy to determine whether a query receives a response with fresh data, or with aged data. In the first case, the query is forwarded to a queue Q_1 where the query is eventually serviced. In the second case, the query is immediately answered with a known, aged, response that is stored in, e.g., a database (DB). The DB is regularly refreshed by reports from a queue Q_2. For modelling purposes we assume that both queries and report requests arrive according to a homogeneous Poisson process with rate λ_1 and λ_2, respectively. Also, we assume that the processing time in the queues is exponentially distributed with parameter μ_1 (for queries) and μ_2 (for report requests).

An example of the interaction between queries, reports, and the age of the latest value in the DB is shown in Figure 5.2. At time 1 a job is completed at the server of Q_2 (resulting in a report) and sets the age to 0. This age then increases linearly until the next report is generated at time 4. Meanwhile, at time 1.5 a query arrives at the controller, at which moment the age of the latest value in the database is 0.5. Then at time 3 the second query arrives, which sees the most recent value in the database at age 2. Query 3 arrives after the second report is generated, at which point the value in the database has age 1. Report 3 at time 6 refreshes the database again and sets the age to 0. Note that the graph does not show which decisions the controller takes on arrival of a query.

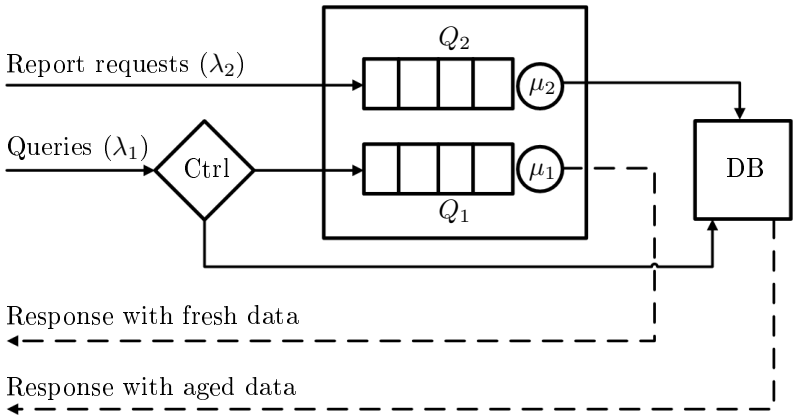


FIGURE 5.1: The controller (Ctrl) assigns incoming queries to either Q_1 in the queueing system, or to the DB. In the first case, the query gets a fresh response, but has to wait some time before it is generated. In the second situation, the systems returns a previously generated (and thus aged) response immediately. The DB is regularly refreshed with fresh values (reports) from Q_2.

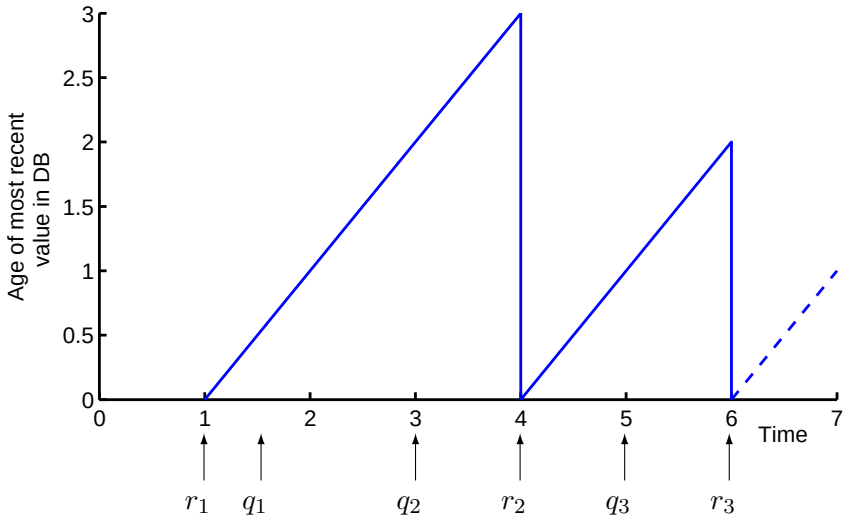


FIGURE 5.2: Interaction between queries $q_1, \ldots, q_3$, reports $r_1, \ldots, r_3$, and the age of the latest value in the database. In the graph, three reports arrive (at times 1, 4, and 6) that reset the age to 0. In between reports, the age increases linearly with time. Upon a query arrival, the controller sees the latest value in the database at a certain age and uses that age to take its decisions. For instance, query q_2 arrives at time 3, at which moment the most recent value in the database has age 2.

The scenario described above is characterized by three distinctive elements: (1) a queueing system, (2) a database that is periodically refreshed from the queueing system, and (3) the controller assigning queries to either of the two other elements. Despite a thorough literature review, we did not find any research with the same combination of elements (apart from [2], where we investigate the same scenario using a different model). Caching scenarios, such as the web server example mentioned before, are related, but seem to be not used together with a queueing system. From a queueing theoretic approach, the papers [41] and [108] are somewhat similar to our situation. They deal with several servers for which aged information about the loads is available and, as in our approach, this aged information is periodically updated by the queues via reports. Their system, however, does not contain a database, but has multiple queues that can serve the incoming jobs. The controller decides which of the queues to use based on the aged load information, and thus addresses a problem different from ours.

Addressing the trade-off between data freshness and response times is traditionally done using a threshold policy. When the age of the database value exceeds a certain given threshold, fresh data is retrieved, and otherwise the latest database value is used. Although such policies are commonly used, there is room for improvement by setting a dynamic threshold: in cases where the information retrieval is time-consuming (as it is in wireless sensor networks), using a database value that is slightly above the threshold value might be acceptable. Motivated by this, our aim in the current chapter is to demonstrate for our example scenario that it is possible to find a policy that performs better than a simple threshold policy. We formulate our control problem as a three-dimensional MDP, which turns out to be complex and hard to solve analytically. Then, we provide a clever strategy for reducing the dimensionality of the model, and construct an approximate model that captures the system dynamics in a simpler way, allowing for an analytical solution. After deriving this solution, we apply one-step policy improvement to obtain an improved policy. We numerically compare this policy to the optimal policy, as well as to a myopic policy and to a traditional age-threshold policy. The improved policy achieves near-optimal performance, and has lower costs than the myopic and the age-threshold policy.

The structure of this chapter is as follows. Section 5.2 introduces the MDP used to model the scenario above, and Section 5.3 illustrates the three steps of our approach to finding a near-optimal control policy. Then, Section 5.4 presents the first of these steps, detailing how the approximate model is constructed. The second step, finding a solution to the approximate model, is in Section 5.5. Section 5.6 contains the third and final step, describing the derivation of our

near-optimal control policy. Numerical experiments with this policy are presented in Section 5.7, as well as a closer look at the optimal policy. We finish with conclusions and future research directions in Section 5.8.

5.2 Model formulation

The trade-off we discuss in this chapter is between data freshness and query response times. Here, we assume that the query response time is proportional to the current workload of the system, i.e., the number of queries plus the number of report requests in the system. The decision (to use either the DB, or the queueing system) thus depends on the number of queries in the system, on the number of report requests, and on the age of the most recent value in the DB. In order to analyze decision policies, we formulate the scenario as an MDP. The state space is $\mathcal{X} = \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0$, where $(i, j, N) \in \mathcal{X}$ denotes a system containing i queries and j report requests, and where the latest report refreshed the DB N time units ago. The controller can choose actions a from $\mathcal{A} = \{Q_1, \text{DB}\}$, where Q_1 indicates forwarding of the query to Q_1 (see Figure 5.1). The cost function $c(i, j, N; a)$ incorporates the costs of each action available to the controller:

$$c(i,j,N;a) = \begin{cases} \gamma_1(i+1) + \gamma_2 j + \gamma_3, & \text{if } a = Q_1, \\ (N-T)^+, & \text{if } a{=}\text{DB}. \end{cases} \tag{5.1}$$

Here, $\gamma_1(i+1) + \gamma_2 j + \gamma_3$ is a weighted sum (with weights $\gamma_1, \gamma_2, \gamma_3 \in \mathbb{R}$) of the number of queries and report requests in the system, reflecting the workload of the system after assigning a new query to it. The term $(N-T)^+$ in the cost function is a penalty for returning a stale value from the DB instead of a fresh value. The parameter T indicates a threshold below which the latest value in the DB is recent enough to answer the query. Note that we took $\gamma_1(i+1)$ rather than $\gamma_1 i$ in the cost function, because we include the query that is about to be assigned to Q_1 when that action is chosen. Additionally, the resulting expression for the improved policy closely resembles a simple myopic policy, enabling us to interpret the difference between the two policies

The state space, the action set, the transition rates, and the cost function define the MDP. More explicitly, the optimality equation of the MDP can be

formulated as follows:

$$\begin{aligned} g + V(i,j,N) = {} & \lambda_2 V(i,j+1,N+1) \\ & + \mu_1 V(i-1,j,N+1)\mathbb{1}_{\{i>0\}} + \mu_2 V(i,j-1,0)\mathbb{1}_{\{j>0\}} \\ & + (1-\lambda_1-\lambda_2-\mu_1\mathbb{1}_{\{i>0\}}-\mu_2\mathbb{1}_{\{j>0\}})V(i,j,N+1) \\ & + \lambda_1 \min\big\{\gamma_1(i+1)+\gamma_2 j+\gamma_3+V(i+1,j,N+1); \\ & \qquad (N-T)^+ + V(i,j,N+1)\big\}, \end{aligned} \tag{5.2}$$

with $V(i,j,N)$ the relative value function and g the time-average costs. The uniformization term is formed by the third line, where we assumed that parameters $\lambda_1, \lambda_2, \mu_1, \mu_2$ are normalized such that $\lambda_1+\lambda_2+\mu_1+\mu_2 = 1$. Hence, we can regard these parameters as transition probabilities and Eq. (5.2) as a discrete-time model. Also note that N measures the number of uniformized time steps since the generation of the last report, and not "real" time. Finally, we assume the stability conditions $\rho_1 := \lambda_1/\mu_1 < 1$ and $\rho_2 := \lambda_2/\mu_2 < 1$ hold.

5.3 Obtaining the near-optimal policy

Ideally, we would like to solve the optimality equation (5.2) analytically and obtain an expression for the relative value function (and, consequently, for the optimal policy). However, the optimality equation has several complicating aspects that prevent us from doing so:

- It contains the decision capturing the trade-off faced by the controller, which involves evaluation of a minimization term.
- In this minimum the inhomogeneous terms $\gamma_1(i+1)+\gamma_2 j+\gamma_3$ and $(N-T)^+$ add to the complexity of the model.
- The state space variables interact with each other, i.e., in Eq. (5.2), $V(i,j,N)$ depends on 'neighbors' $V(i,j+1,N+1)$, $V(i-1,j,N+1)$, $V(i,j,N+1)$, and $V(i+1,j,N+1)$, so it is challenging to find a solution by decomposing the state space.
- A complex relation between j and N exists due to the term $\mu_2 V(i,j-1,0)$.

In order to circumvent these complexities, we take three steps in the upcoming sections. We derive an approximate model to the original problem (step I), which we can solve analytically for a specific policy (step II). This solution is then converted into a near-optimal control policy for the original problem (step III). In more detail, the three steps are:

I We start in Section 5.4 with a modification of the optimality equation (5.2), obtained by removing the N-dimension, resulting in an MDP for an approximation to $V(i,j,N)$ (denoted by $\widetilde{V}(i,j)$).

II In Section 5.5 we choose a policy for this new MDP and solve it analytically, yielding a solution $\widetilde{V}^{\alpha}(i,j)$. Here, α is the parameter of a Bernoulli routing policy.

III Finally, in Section 5.6, we apply *one-step policy improvement* by inspecting the minimum in Eq. (5.2), substituting $\widetilde{V}^{\alpha}(i,j)$ for $V(i,j,N)$. This results in an improved policy, denoted by π'.

Eq. (5.3) schematically reflects the steps and the notation used.

$$V(i,j,N) \xrightarrow{I} \widetilde{V}(i,j) \xrightarrow{II} \widetilde{V}^{\alpha}(i,j) \xrightarrow{III} \pi'. \tag{5.3}$$

5.4 Step I: model approximation

Looking at Eq. (5.2), we see that N is in the state space to accommodate the penalty term $(N-T)^+$. Therefore, if we replace the $(N-T)^+$ by a suitable constant C, the N can be removed from the state space. Introducing the constant C in Eq. (5.2) yields

$$\begin{aligned}\tilde{g} + \widetilde{V}(i,j) = {} & \lambda_2 \widetilde{V}(i,j+1) \\ & + \mu_1 \widetilde{V}(i-1,j)\mathbb{1}_{\{i>0\}} + \mu_2 \widetilde{V}(i,j-1)\mathbb{1}_{\{j>0\}} \\ & + (1-\lambda_1-\lambda_2-\mu_1\mathbb{1}_{\{i>0\}}-\mu_2\mathbb{1}_{\{j>0\}})\widetilde{V}(i,j) \\ & + \lambda_1 \min\left\{\gamma_1(i+1)+\gamma_2 j+\gamma_3+\widetilde{V}(i+1,j); C+\widetilde{V}(i,j)\right\}.\end{aligned} \tag{5.4}$$

As it turns out, the constant C does not affect our near-optimal policy, so assigning a value to it is not strictly necessary (in Section 5.6.1 the term $(N-T)^+$ is reintroduced). However, the idea of reducing the state space in this manner might be applicable to other MDPs, so for completeness we shortly illustrate how C can be determined for Eq. (5.2). To this end, we inspect this MDP for the policy that always uses the DB to answer queries. Replacing the minimum in Eq. (5.2) by this policy yields the equation

$$\begin{aligned}g^{DB} + V^{DB}(j,N) = {} & \lambda_2 V^{DB}(j+1,N+1) + \mu_2 V^{DB}(j-1,0)\mathbb{1}_{\{j>0\}} \\ & + (1-\lambda_1-\lambda_2-\mu_2\mathbb{1}_{\{j>0\}})V^{DB}(j,N+1) \\ & + \lambda_1\left((N-T)^+ + V^{DB}(j,N+1)\right),\end{aligned} \tag{5.5}$$

where variable i is removed from the notation because it no longer influences the relative value function. In [2, Appendix B], we show that $g^{DB} = \lambda_1 \frac{(1-\lambda_2)^{T+1}}{\lambda_2}$. Note that if we replace $(N-T)^+$ by constant C in Eq. (5.5), then we would have $g^{DB} = \lambda_1 C$. A suitable choice for C is one that leaves the time-average costs intact, i.e., $C = \frac{(1-\lambda_2)^{T+1}}{\lambda_2}$.

5.5 Step II: near-optimal control policies

The next step is to fix a policy for the MDP in Eq. (5.4) so that we can obtain an analytic expression for the corresponding relative value function. For this policy we choose the *Bernoulli policy*, which randomly assigns incoming queries to either Q_1 (with probability $\alpha \in [0,1]$) or to the DB (with probability $1-\alpha$). Replacing the minimum in Eq. (5.4) by the Bernoulli policy yields the difference equation

$$\begin{aligned}\tilde{g}^\alpha + \widetilde{V}^\alpha(i,j) = {} & \lambda_2 \widetilde{V}^\alpha(i,j+1) \\ & + \mu_1 \widetilde{V}^\alpha(i-1,j)\mathbb{1}_{\{i>0\}} + \mu_2 \widetilde{V}^\alpha(i,j-1)\mathbb{1}_{\{j>0\}} \\ & + (1-\lambda_1-\lambda_2-\mu_1\mathbb{1}_{\{i>0\}}-\mu_2\mathbb{1}_{\{j>0\}})\widetilde{V}^\alpha(i,j) \\ & + \lambda_1\alpha\left[\gamma_1(i+1)+\gamma_2 j+\gamma_3+\widetilde{V}^\alpha(i+1,j)\right] \\ & + \lambda_1(1-\alpha)\left[C+\widetilde{V}^\alpha(i,j)\right].\end{aligned} \tag{5.6}$$

Note how the application of the Bernoulli policy decouples the queueing system from the DB. In the remainder of this section we derive an expression for the relative value function $\widetilde{V}^\alpha(i,j)$ by solving Eq. (5.6). This result is summarized in the following theorem:

THEOREM 5.5.1. *The solution to Eq. (5.6) is given by*

$$\widetilde{V}^\alpha(i,j) = \frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha}\frac{i(i+1)}{2} + \frac{\gamma_2\lambda_1\alpha}{\mu_2-\lambda_2}\frac{j(j+1)}{2},$$

and

$$\tilde{g}^\alpha = \lambda_1(1-\alpha)C + \lambda_1\alpha\left(\frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha} + \frac{\gamma_2\lambda_2}{\mu_2-\lambda_2} + \gamma_1 + \gamma_3\right).$$

Substitution of these expressions for $\widetilde{V}^\alpha(i,j)$ and $\tilde{g}^\alpha$ into Eq. (5.6) shows that these indeed form a solution. In the following subsections we derive

the expressions in Theorem 5.5.1 by solving Eq. (5.6). First, we tackle the inhomogeneous terms $\gamma_1(i+1)+\gamma_2 j+\gamma_3$ and C by considering an equation for $\Delta_1\widetilde{V}^\alpha(i,j) = \widetilde{V}^\alpha(i+1,j) - \widetilde{V}^\alpha(i,j)$. This removes the inhomogeneous term C and transforms the other term to γ_1. Then we look at $\Delta_1^2\widetilde{V}^\alpha(i,j) = \Delta_1\widetilde{V}^\alpha(i+1,j) - \Delta_1\widetilde{V}^\alpha(i,j)$, which eliminates the remaining inhomogeneous term γ_1. We solve this equation, and then retrace our steps from $\Delta_1^2\widetilde{V}^\alpha(i,j)$ to $\Delta_1\widetilde{V}^\alpha(i,j)$ to $\widetilde{V}^\alpha(i,j)$.

During the derivation we encounter an issue concerning uniqueness of solutions to the Poisson equation for $\Delta_1^2\widetilde{V}^\alpha(i,j)$. There, we postulate a form for a solution and must show that this solution is unique. Showing uniqueness is not trivial and involves several technical arguments that result in additional restrictions on the form of $\Delta_1^2\widetilde{V}^\alpha(i,j)$. This important part of the derivation is placed in Section 5.9.

5.5.1 Solving the difference equation for $\Delta_1^2\widetilde{V}^\alpha(i,j)$

The behavior of the difference equation on the interior of the state space differs from the behavior on the boundaries $\{i=0\}$ and $\{j=0\}$. Therefore, we first study the difference equation for the interior $\{i,j>0\}$ of the state space. We define $\Delta_1\widetilde{V}^\alpha(i,j) := \widetilde{V}^\alpha(i+1,j) - \widetilde{V}^\alpha(i,j)$, and for $i>0$ and $j>0$ it holds that

$$\begin{aligned}\Delta_1\widetilde{V}^\alpha(i,j) = {}& \lambda_1\alpha\Big[\gamma_1 + \Delta_1\widetilde{V}^\alpha(i+1,j)\Big] + \lambda_1(1-\alpha)\Delta_1\widetilde{V}^\alpha(i,j)\\ &+ \lambda_2\Delta_1\widetilde{V}^\alpha(i,j+1)\\ &+ \mu_1\Delta_1\widetilde{V}^\alpha(i-1,j) + \mu_2\Delta_1\widetilde{V}^\alpha(i,j-1)\\ &+ (1-\lambda_1-\lambda_2-\mu_1-\mu_2)\Delta_1\widetilde{V}^\alpha(i,j).\end{aligned} \tag{5.7}$$

Now, define $\Delta_1^2\widetilde{V}^\alpha(i,j) = \Delta_1\widetilde{V}^\alpha(i+1,j) - \Delta_1\widetilde{V}^\alpha(i,j)$, for which we have

$$\begin{aligned}\Delta_1^2\widetilde{V}^\alpha(i,j) = {}& \lambda_1\alpha\Delta_1^2\widetilde{V}^\alpha(i+1,j) + \lambda_1(1-\alpha)\Delta_1^2\widetilde{V}^\alpha(i,j)\\ &+ \lambda_2\Delta_1^2\widetilde{V}^\alpha(i,j+1)\\ &+ \mu_1\Delta_1^2\widetilde{V}^\alpha(i-1,j) + \mu_2\Delta_1^2\widetilde{V}^\alpha(i,j-1)\\ &+ (1-\lambda_1-\lambda_2-\mu_1-\mu_2)\Delta_1^2\widetilde{V}^\alpha(i,j).\end{aligned}$$

We suggestively write this as

$$\begin{aligned}(\lambda_1\alpha+\mu_1)\Delta_1^2\widetilde{V}^\alpha(i,j) + (\lambda_2+\mu_2)\Delta_1^2\widetilde{V}^\alpha(i,j) = {}& \\ \lambda_1\alpha\Delta_1^2\widetilde{V}^\alpha(i+1,j) + \mu_1\Delta_1^2\widetilde{V}^\alpha(i-1,j) & \\ + \lambda_2\Delta_1^2\widetilde{V}^\alpha(i,j+1) + \mu_2\Delta_1^2\widetilde{V}^\alpha(i,j-1). &\end{aligned} \tag{5.8}$$

The notation suggests that the solution to this equation might be split up in a part that only depends on i and a part that only depends on j. That is, a solution might be given by $\Delta_1^2\widetilde{V}^\alpha(i,j) = \widetilde{V}_1^\alpha(i) + \widetilde{V}_2^\alpha(j)$ with $\widetilde{V}_1^\alpha(i)$ and $\widetilde{V}_2^\alpha(j)$ satisfying

$$\begin{cases} (\lambda_1\alpha + \mu_1)\widetilde{V}_1^\alpha(i) = \lambda_1\alpha\widetilde{V}_1^\alpha(i+1) + \mu_1\widetilde{V}_1^\alpha(i-1), \\ (\lambda_2 + \mu_2)\widetilde{V}_2^\alpha(j) = \lambda_2\widetilde{V}_2^\alpha(j+1) + \mu_2\widetilde{V}_2^\alpha(j-1). \end{cases} \tag{5.9}$$

These equations are simple homogeneous difference equations of which the solutions are given by

$$\begin{cases} \widetilde{V}_1^\alpha(i) = \dfrac{\mu_1\widetilde{V}_1^\alpha(0) - \lambda_1\alpha\widetilde{V}_1^\alpha(1)}{\mu_1 - \lambda_1\alpha} + \dfrac{\lambda_1\alpha\big(\widetilde{V}_1^\alpha(1) - \widetilde{V}_1^\alpha(0)\big)\left(\frac{\mu_1}{\lambda_1\alpha}\right)^i}{\mu_1 - \lambda_1\alpha}, \\ \widetilde{V}_2^\alpha(j) = \dfrac{\mu_2\widetilde{V}_2^\alpha(0) - \lambda_2\widetilde{V}_2^\alpha(1)}{\mu_2 - \lambda_2} + \dfrac{\lambda_2\big(\widetilde{V}_2^\alpha(1) - \widetilde{V}_2^\alpha(0)\big)\left(\frac{\mu_2}{\lambda_2}\right)^j}{\mu_2 - \lambda_2}. \end{cases} \tag{5.10}$$

Note that with these expressions for $\widetilde{V}_1\alpha(i)$ and $\widetilde{V}_2\alpha(j)$, $\Delta_1^2\widetilde{V}^\alpha(i,j)$ is **a** solution to Eq. (5.8). It is, however, not immediately obvious that this is also **the** solution. We return to this issue in Section 5.9.

The values for $\widetilde{V}_1^\alpha(0), \widetilde{V}_1^\alpha(1), \widetilde{V}_2^\alpha(0), \widetilde{V}_2^\alpha(1)$ still need to be determined to make the solution consistent at the boundaries. For this purpose, consider the boundary $\{j = 0\}$ of the state space, where $\Delta_1\widetilde{V}^\alpha(i,0)$ becomes (for $i > 0$)

$$\begin{aligned} \Delta_1\widetilde{V}^\alpha(i,0) = {} & \lambda_1\alpha\Big[\gamma_1 + \Delta_1\widetilde{V}^\alpha(i+1,0)\Big] + \lambda_1(1-\alpha)\Delta_1\widetilde{V}^\alpha(i,0) \\ & + \lambda_2\Delta_1\widetilde{V}^\alpha(i,1) + \mu_1\Delta_1\widetilde{V}^\alpha(i-1,0) \\ & + (1 - \lambda_1 - \lambda_2 - \mu_1)\Delta_1\widetilde{V}^\alpha(i,0). \end{aligned} \tag{5.11}$$

Similarly, for $\Delta_1^2\widetilde{V}^\alpha(i,0)$ we have that

$$\begin{aligned} \Delta_1^2\widetilde{V}^\alpha(i,0) = {} & \lambda_1\alpha\Delta_1^2\widetilde{V}^\alpha(i+1,0) + \lambda_1(1-\alpha)\Delta_1^2\widetilde{V}^\alpha(i,0) \\ & + \lambda_2\Delta_1^2\widetilde{V}^\alpha(i,1) + \mu_1\Delta_1^2\widetilde{V}^\alpha(i-1,0) \\ & + (1 - \lambda_1 - \lambda_2 - \mu_1)\Delta_1^2\widetilde{V}^\alpha(i,0). \end{aligned}$$

Again, we can suggestively write this as

$$\begin{aligned} (\lambda_1\alpha + \mu_1)\Delta_1^2\widetilde{V}^\alpha(i,0) + \lambda_2\Delta_1^2\widetilde{V}^\alpha(i,0) = {} & \\ \lambda_1\alpha\Delta_1^2\widetilde{V}^\alpha(i+1,0) + \mu_1\Delta_1^2\widetilde{V}^\alpha(i-1,0) & \\ + \lambda_2\Delta_1^2\widetilde{V}^\alpha(i,1), & \end{aligned}$$

leading to the following system of equations

$$\begin{cases} (\lambda_1\alpha+\mu_1)\widetilde{V}_1^\alpha(i)=\lambda_1\alpha\widetilde{V}_1^\alpha(i+1)+\mu_1\widetilde{V}_1^\alpha(i-1), \\ \qquad\qquad \lambda_2\widetilde{V}_2^\alpha(0)=\lambda_2\widetilde{V}_2^\alpha(1). \end{cases} \tag{5.12}$$

From these expressions, we obtain that on the boundary $\{j=0\}$ of the state space, the MDP behaves exactly the same as the MDP on the interior of the state space. Furthermore, it shows that $\widetilde{V}_2^\alpha(0)=\widetilde{V}_2^\alpha(1)$ and thus that $\widetilde{V}_2^\alpha(j)$ in Eq. (5.10) is a constant: $\widetilde{V}_2^\alpha(j)=c_2$. Without loss of generality, we can set $c_2=0$ and determine $\Delta_1^2\widetilde{V}^\alpha(i,j)$ completely from $\widetilde{V}_1^\alpha(i)$. Hence, we have $\Delta_1^2\widetilde{V}^\alpha(i,j)=\widetilde{V}_1^\alpha(i)+\widetilde{V}_2^\alpha(j)$, where $\widetilde{V}_2^\alpha(j)\equiv 0$ and

$$\widetilde{V}_1^\alpha(i)=\frac{\mu_1\widetilde{V}_1^\alpha(0)-\lambda_1\alpha\widetilde{V}_1^\alpha(1)}{\mu_1-\lambda_1\alpha}+\frac{\lambda_1\alpha\big(\widetilde{V}_1^\alpha(1)-\widetilde{V}_1^\alpha(0)\big)\left(\frac{\mu_1}{\lambda_1\alpha}\right)^i}{\mu_1-\lambda_1\alpha}.$$

5.5.2 Analyzing $\Delta_1\widetilde{V}^\alpha(i,j+1)-\Delta_1\widetilde{V}^\alpha(i,j)$

For the derivation of an expression for $\widetilde{V}^\alpha(i,j)$ (which we do in the next sections), we require an intermediate result about $\Delta_1\widetilde{V}^\alpha(i,j+1)-\Delta_1\widetilde{V}^\alpha(i,j)$. With notation

$$\Delta_2\Delta_1\widetilde{V}^\alpha(i,j):=\Delta_1\widetilde{V}^\alpha(i,j+1)-\Delta_1\widetilde{V}^\alpha(i,j),$$

we prove the following lemma in this section:

LEMMA 5.5.2. *The relative value function $\widetilde{V}^\alpha(i,j)$ satisfies*

$$\Delta_2\Delta_1\widetilde{V}^\alpha(i,j)=0.$$

In words, Lemma 5.5.2 states that first differencing $\widetilde{V}^\alpha(i,j)$ in i, followed by differencing the result in j, equals 0.

PROOF. We start again for the interior $\{i,j>0\}$ of the state space, where we have the following relation for $i>0$ and $j>0$:

$$\begin{aligned}\Delta_1\widetilde{V}^\alpha(i,j)=\lambda_1\alpha\Big[\gamma_1+\Delta_1\widetilde{V}^\alpha(i+1,j)\Big]&+\lambda_1(1-\alpha)\Delta_1\widetilde{V}^\alpha(i,j)\\ &+\lambda_2\Delta_1\widetilde{V}^\alpha(i,j+1)\\ &+\mu_1\Delta_1\widetilde{V}^\alpha(i-1,j)+\mu_2\Delta_1\widetilde{V}^\alpha(i,j-1)\\ &+(1-\lambda_1-\lambda_2-\mu_1-\mu_2)\Delta_1\widetilde{V}^\alpha(i,j).\end{aligned}$$

We find for $\Delta_2\Delta_1\widetilde{V}^\alpha(i,j)$

$$\begin{aligned}\Delta_2\Delta_1\widetilde{V}^\alpha(i,j) &= \lambda_1\alpha\Delta_2\Delta_1\widetilde{V}^\alpha(i+1,j) + \lambda_1(1-\alpha)\Delta_2\Delta_1\widetilde{V}^\alpha(i,j)\\ &\quad + \lambda_2\Delta_2\Delta_1\widetilde{V}^\alpha(i,j+1)\\ &\quad + \mu_1\Delta_2\Delta_1\widetilde{V}^\alpha(i-1,j) + \mu_2\Delta_2\Delta_1\widetilde{V}^\alpha(i,j-1)\\ &\quad + (1-\lambda_1-\lambda_2-\mu_1-\mu_2)\Delta_2\Delta_1\widetilde{V}^\alpha(i,j).\end{aligned} \tag{5.13}$$

By similar line of reasoning as before, we derive that $\Delta_2\Delta_1\widetilde{V}^\alpha(i,j) = \bar{V}_1(i) + \bar{V}_2(j)$, with

$$\begin{aligned}\bar{V}_1(i) &= \frac{\mu_1\bar{V}_1(0) - \lambda_1\alpha\bar{V}_1(1)}{\mu_1-\lambda_1\alpha} + \frac{\lambda_1\alpha\big(\bar{V}_1(1)-\bar{V}_1(0)\big)\left(\frac{\mu_1}{\lambda_1\alpha}\right)^i}{\mu_1-\lambda_1\alpha},\\ \bar{V}_2(j) &= \frac{\mu_2\bar{V}_2(0) - \lambda_2\bar{V}_2(1)}{\mu_2-\lambda_2} + \frac{\lambda_2\big(\bar{V}_2(1)-\bar{V}_2(0)\big)\left(\frac{\mu_2}{\lambda_2}\right)^j}{\mu_2-\lambda_2},\end{aligned} \tag{5.14}$$

where the values for $\bar{V}_1(0), \bar{V}_1(1), \bar{V}_2(0), \bar{V}_2(1)$ are determined from $\Delta_2\Delta_1\widetilde{V}^\alpha(i,0)$ and $\Delta_2\Delta_1\widetilde{V}^\alpha(0,j)$. We start with the former by inspecting the term $\Delta_1\widetilde{V}^\alpha(i,1)$. From Eq. (5.7) we have that

$$\begin{aligned}\Delta_1\widetilde{V}^\alpha(i,1) &= \lambda_1\alpha\Big[\gamma_1 + \Delta_1\widetilde{V}^\alpha(i+1,1)\Big] + \lambda_1(1-\alpha)\Delta_1\widetilde{V}^\alpha(i,1)\\ &\quad + \lambda_2\Delta_1\widetilde{V}^\alpha(i,2)\\ &\quad + \mu_1\Delta_1\widetilde{V}^\alpha(i-1,1) + \mu_2\Delta_1\widetilde{V}^\alpha(i,0)\\ &\quad + (1-\lambda_1-\lambda_2-\mu_1-\mu_2)\Delta_1\widetilde{V}^\alpha(i,1).\end{aligned}$$

The term $\Delta_1\widetilde{V}^\alpha(i,0)$ can be obtained from Eq. (5.11), which we repeat here for convenience:

$$\begin{aligned}\Delta_1\widetilde{V}^\alpha(i,0) &= \lambda_1\alpha\Big[\gamma_1 + \Delta_1\widetilde{V}^\alpha(i+1,0)\Big] + \lambda_1(1-\alpha)\Delta_1\widetilde{V}^\alpha(i,0)\\ &\quad + \lambda_2\Delta_1\widetilde{V}^\alpha(i,1)\\ &\quad + \mu_1\Delta_1\widetilde{V}^\alpha(i-1,0)\\ &\quad + (1-\lambda_1-\lambda_2-\mu_1)\Delta_1\widetilde{V}^\alpha(i,0).\end{aligned}$$

Consequently,

$$\begin{aligned}\Delta_2\Delta_1\widetilde{V}^\alpha(i,0) &= \lambda_1\alpha\Delta_2\Delta_1\widetilde{V}^\alpha(i+1,0) + \lambda_1(1-\alpha)\Delta_2\Delta_1\widetilde{V}^\alpha(i,0)\\ &\quad + \lambda_2\Delta_2\Delta_1\widetilde{V}^\alpha(i,1)\\ &\quad + \mu_1\Delta_2\Delta_1\widetilde{V}^\alpha(i-1,0) - \mu_2\Delta_2\Delta_1\widetilde{V}^\alpha(i,0)\\ &\quad + (1-\lambda_1-\lambda_2-\mu_1)\Delta_2\Delta_1\widetilde{V}^\alpha(i,0),\end{aligned}$$

which reduces to

$$(\lambda_2+\mu_2)\Delta_2\Delta_1\widetilde{V}^\alpha(i,0)+(\lambda_1\alpha+\mu_1)\Delta_2\Delta_1\widetilde{V}^\alpha(i,0)= \\ \lambda_2\Delta_2\Delta_1\widetilde{V}^\alpha(i,1)+\lambda_1\alpha\Delta_2\Delta_1\widetilde{V}^\alpha(i+1,0)+\mu_1\Delta_2\Delta_1\widetilde{V}^\alpha(i-1,0).$$

Again we propose a solution of the type $\Delta_2\Delta_1\widetilde{V}^\alpha(i,j)=\bar{V}_1(i)+\bar{V}_2(j)$, resulting in

$$\begin{cases} (\lambda_2+\mu_2)\big(\bar{V}_1(i)+\bar{V}_2(0)\big)=\lambda_2\big(\bar{V}_1(i)+\bar{V}_2(1)\big), \\ (\lambda_1\alpha+\mu_1)(\bar{V}_1(i)+\bar{V}_2(0))=\lambda_1\alpha(\bar{V}_1(i+1)+\bar{V}_2(0)) \\ \qquad\qquad\qquad\qquad +\mu_1(\bar{V}_1(i-1)+\bar{V}_2(0)). \end{cases} \tag{5.15}$$

The upper equation translates to

$$\mu_2\bar{V}_1(i)=\lambda_2\bar{V}_2(1)-(\lambda_2+\mu_2)\bar{V}_2(0), \tag{5.16}$$

i.e., $\bar{V}_1(i)$ is constant for $i>0$, which we denote by $\bar{V}_1(i)=\bar{c}_1$. By repeating the arguments above for the boundary $\{i=0\}$ of the state space, we find that $\bar{V}_2(j):=\bar{c}_2$ is constant. As a consequence, Eq. (5.16) reduces to

$$\mu_2\bar{c}_1=\lambda_2\bar{c}_2-(\lambda_2+\mu_2)\bar{c}_2,$$

or

$$\mu_2\bar{c}_1=-\mu_2\bar{c}_2,$$

i.e., $\bar{c}_1=-\bar{c}_2$ and thus $\Delta_2\Delta_1\widetilde{V}^\alpha(i,j)=0$, which concludes the proof.

□

5.5.3 Solving the difference equation for $\Delta_1\widetilde{V}^\alpha(i,j)$

So far, we have found that $\Delta_1^2\widetilde{V}^\alpha(i,j)$ satisfies

$$\Delta_1^2\widetilde{V}^\alpha(i,j)=\frac{\mu_1\widetilde{V}_1^\alpha(0)-\lambda_1\alpha\widetilde{V}_1^\alpha(1)}{\mu_1-\lambda_1\alpha}+\frac{\lambda_1\alpha\big(\widetilde{V}_1^\alpha(1)-\widetilde{V}_1^\alpha(0)\big)\left(\frac{\mu_1}{\lambda_1\alpha}\right)^i}{\mu_1-\lambda_1\alpha}, \tag{5.17}$$

and we proved that $\Delta_2\Delta_1\widetilde{V}^\alpha(i,j)=0$ in Lemma 5.5.2. Note that this implies that $\Delta_1\widetilde{V}^\alpha(i,j)$ is independent of j for all i. We continue the proof of Theorem. 5.5.1 by reverting the differencing in i used to obtain Eq. (5.17).

Recall that $\Delta_1^2\widetilde{V}^\alpha(i,j) = \Delta_1\widetilde{V}^\alpha(i+1,j) - \Delta_1\widetilde{V}^\alpha(i,j)$. By summing over i, and then using the right-hand side of Eq. (5.17), we can get an expression for $\Delta_1\widetilde{V}^\alpha(i,j)$:

$$\begin{aligned}\Delta_1\widetilde{V}^\alpha(i,j) &= \Delta_1\widetilde{V}^\alpha(0,j) + \sum_{k=0}^{i-1}\Delta_1^2\widetilde{V}^\alpha(k,j)\\ &= \Delta_1\widetilde{V}^\alpha(0,j) + \frac{\mu_1\widetilde{V}_1^\alpha(0) - \lambda_1\alpha\widetilde{V}_1^\alpha(1)}{\mu_1-\lambda_1\alpha}i\\ &\quad + \frac{\lambda_1\alpha\big(\widetilde{V}_1^\alpha(1) - \widetilde{V}_1^\alpha(0)\big)}{\mu_1-\lambda_1\alpha}\cdot\frac{1-(\frac{\mu_1}{\lambda_1\alpha})^i}{1-\frac{\mu_1}{\lambda_1\alpha}}.\end{aligned} \tag{5.18}$$

Here, $\Delta_1\widetilde{V}^\alpha(0,j)$ is a constant (by Lemma 5.5.2) which we determine below. Substituting the expression for $\Delta_1\widetilde{V}^\alpha(i,j)$ from Eq. (5.18) into Eq. (5.7), we find that necessarily

$$\mu_1\widetilde{V}_1^\alpha(0) - \lambda_1\alpha\widetilde{V}_1^\alpha(1) = \gamma_1\lambda_1\alpha.$$

Solving this for $\widetilde{V}_1^\alpha(1)$ and substituting the result into Eq. (5.17) yields

$$\Delta_1^2\widetilde{V}^\alpha(i,j) = \frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha} + \left[\widetilde{V}_1^\alpha(0) - \frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha}\right]\left(\frac{\mu_1}{\lambda_1\alpha}\right)^i.$$

Hence, $\Delta_1\widetilde{V}^\alpha(i,j)$ becomes

$$\begin{aligned}\Delta_1\widetilde{V}^\alpha(i,j) &= \Delta_1\widetilde{V}^\alpha(0,j) + \frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha}i\\ &\quad + \left[\widetilde{V}_1^\alpha(0) - \frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha}\right]\frac{1-(\frac{\mu_1}{\lambda_1\alpha})^i}{1-\frac{\mu_1}{\lambda_1\alpha}}.\end{aligned} \tag{5.19}$$

Now we turn our attention to determining the (constant) $\Delta_1\widetilde{V}^\alpha(0,j)$ by inspecting the corresponding difference equation:

$$\begin{aligned}\Delta_1\widetilde{V}^\alpha(0,j) &= \lambda_1\alpha\Big[\gamma_1 + \Delta_1\widetilde{V}^\alpha(1,j)\Big] + \lambda_1(1-\alpha)\Delta_1\widetilde{V}^\alpha(0,j)\\ &\quad + \lambda_2\Delta_1\widetilde{V}^\alpha(0,j+1) + \mu_2\Delta_1\widetilde{V}^\alpha(0,j-1)\\ &\quad + (1-\lambda_1-\lambda_2-\mu_1-\mu_2)\Delta_1\widetilde{V}^\alpha(0,j).\end{aligned}$$

We can rewrite this equation as follows:

$$\begin{aligned}0 &= \lambda_1\alpha[\Delta_1\widetilde{V}^\alpha(1,j) - \Delta_1\widetilde{V}^\alpha(0,j)] + \gamma_1\lambda_1\alpha\\ &\quad + \lambda_2[\Delta_1\widetilde{V}^\alpha(0,j+1) - \Delta_1\widetilde{V}^\alpha(0,j)]\\ &\quad + \mu_2[\Delta_1\widetilde{V}^\alpha(0,j-1) - \Delta_1\widetilde{V}^\alpha(0,j)] - \mu_1\Delta_1\widetilde{V}^\alpha(0,j).\end{aligned}$$

Using Lemma 5.5.2 we find

$$0 = \lambda_1\alpha[\Delta_1\widetilde{V}^\alpha(1,j) - \Delta_1\widetilde{V}^\alpha(0,j)] + \gamma_1\lambda_1\alpha - \mu_1\Delta_1\widetilde{V}^\alpha(0,j).$$

Eq. (5.19) tells us that $\Delta_1\widetilde{V}^\alpha(1,j) = \Delta_1\widetilde{V}^\alpha(0,j) + \widetilde{V}_1^\alpha(0)$, so

$$\Delta_1\widetilde{V}^\alpha(0,j) = \frac{\lambda_1\alpha}{\mu_1}\widetilde{V}_1^\alpha(0) + \frac{\gamma_1\lambda_1\alpha}{\mu_1}.$$

Substitution into Eq. (5.19) yields

$$\begin{aligned}\Delta_1\widetilde{V}^\alpha(i,j) = {} & \frac{\lambda_1\alpha}{\mu_1}\widetilde{V}_1^\alpha(0) + \frac{\gamma_1\lambda_1\alpha}{\mu_1} + \frac{\gamma_1\lambda_1\alpha}{\mu_1 - \lambda_1\alpha}i \\ & + \left[\widetilde{V}_1^\alpha(0) - \frac{\gamma_1\lambda_1\alpha}{\mu_1 - \lambda_1\alpha}\right]\frac{1 - (\frac{\mu_1}{\lambda_1\alpha})^i}{1 - \frac{\mu_1}{\lambda_1\alpha}}.\end{aligned} \tag{5.20}$$

5.5.4 Deriving $\widetilde{V}^\alpha(i,j)$

We derive an expression for $\widetilde{V}^\alpha(i,j)$ using $\Delta_1\widetilde{V}^\alpha(i,j) = \widetilde{V}^\alpha(i+1,j) - \widetilde{V}^\alpha(i,j)$, then summing over i, followed by applying Eq. (5.20):

$$\begin{aligned}\widetilde{V}^\alpha(i,j) &= \widetilde{V}^\alpha(0,j) + \sum_{k=0}^{i-1}\Delta_1\widetilde{V}^\alpha(k,j) \\ &= \widetilde{V}^\alpha(0,j) + i\Big(\frac{\lambda_1\alpha}{\mu_1}\widetilde{V}_1^\alpha(0) + \frac{\gamma_1\lambda_1\alpha}{\mu_1}\Big) \\ &\quad + \frac{\gamma_1\lambda_1\alpha}{\mu_1 - \lambda_1\alpha}\frac{i(i-1)}{2} \\ &\quad + \frac{1}{1 - \frac{\mu_1}{\lambda_1\alpha}}\left[\widetilde{V}_1^\alpha(0) - \frac{\gamma_1\lambda_1\alpha}{\mu_1 - \lambda_1\alpha}\right]\left[i - \frac{1 - (\frac{\mu_1}{\lambda_1\alpha})^i}{1 - \frac{\mu_1}{\lambda_1\alpha}}\right].\end{aligned} \tag{5.21}$$

In the derivation so far we have postulated a form of a solution several times (Eqs. (5.9) and (5.12–5.15)), resulting in the expression for $\widetilde{V}^\alpha(i,j)$ in Eq. (5.21). Here, we finally deal with the uniqueness issue. As mentioned earlier, ensuring uniqueness of a solution $\widetilde{V}^\alpha(i,j)$ to Eq. (5.6) is not trivial. Conventional uniqueness proofs rely on bounded cost functions, and the cost function in Eq. (5.1) is unbounded. Addressing this point requires several technical arguments which we, for readability, place in Section 5.9. In short, uniqueness is ensured if $\widetilde{V}^\alpha(i,j)$ does not grow exponentially fast. Therefore, we choose the remaining constant $\widetilde{V}_1^\alpha(0)$ in Eq. (5.21) such that the exponential term $(\frac{\mu_1}{\lambda_1\alpha})^i$ disappears:

$$\widetilde{V}_1^\alpha(0) = \frac{\gamma_1\lambda_1\alpha}{\mu_1 - \lambda_1\alpha}.$$

Substitution into Eq. (5.21) yields

$$\widetilde{V}^{\alpha}(i,j) = \widetilde{V}^{\alpha}(0,j) + \frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha}i + \frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha}\frac{i(i-1)}{2},$$

or

$$\widetilde{V}^{\alpha}(i,j) = \widetilde{V}^{\alpha}(0,j) + \frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha}\frac{i(i+1)}{2}.$$

Repeating the steps in Sections 5.5.1-5.5.4 for differencing in j instead of i gives

$$\widetilde{V}^{\alpha}(i,j) = \widetilde{V}^{\alpha}(i,0) + \frac{\gamma_2\lambda_1\alpha}{\mu_2-\lambda_2}\frac{j(j+1)}{2},$$

so that necessarily

$$\widetilde{V}^{\alpha}(i,j) = \frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha}\frac{i(i+1)}{2} + \frac{\gamma_2\lambda_1\alpha}{\mu_2-\lambda_2}\frac{j(j+1)}{2}. \tag{5.22}$$

Finally, substituting this expression for $\widetilde{V}^{\alpha}(i,j)$ into Eq. (5.6) and solving for $\tilde{g}^{\alpha}$ yields

$$\tilde{g}^{\alpha} = \lambda_1(1-\alpha)C + \lambda_1\alpha\left(\frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha} + \frac{\gamma_2\lambda_2}{\mu_2-\lambda_2} + \gamma_1 + \gamma_3\right). \tag{5.23}$$

This concludes the derivation of the expressions in Theorem 5.5.1.

Remark: The structure of $\widetilde{V}^{\alpha}(i,j)$ in Eq. (5.22) and $\tilde{g}^{\alpha}$ in Eq. (5.23) can be explained intuitively using known results about the $M/M/1$ queue. The Bernoulli policy chooses Q_1 with probability α and the DB with probability $1-\alpha$, thereby decoupling the system in three separate elements: the DB, Q_1, and Q_2. Choosing the DB incurs a penalty C, which results in time-average costs $\lambda_1(1-\alpha)C$. This corresponds to the first term in Eq. (5.23). The alternative choice (assignment to the queueing system) incurs costs $\gamma_1(i+1) + \gamma_2 j + \gamma_3$. Note that the two queues (the first with arrival rate $\lambda_1\alpha$, the second with arrival rate λ_2) are independent and that the i and j terms are summed in the cost function. Consequently, the time-average costs of assignment to the queueing system are just the summed time-average costs of the two $M/M/1$ queues with holding costs γ_1 and γ_2 respectively (and of fixed costs $\gamma_1+\gamma_3$). For a $M/M/1$ queue we know (from, e.g., [28]) that $g = \frac{\rho}{1-\rho}h$, with $\rho = \lambda/\mu$ the system load, λ the arrival rate, μ the service rate, and holding costs h. This explains the $\frac{\gamma_1\lambda_1\alpha}{\mu_1-\lambda_1\alpha} + \frac{\gamma_2\lambda_2}{\mu_2-\lambda_2} + \gamma_1 + \gamma_3$ term (multiplied by $\lambda_1\alpha$) in Eq. (5.23). Also, the relative value function $\widetilde{V}^{\alpha}(i,j)$ in Eq. (5.22) is just the sum of the relative value functions of the two $M/M/1$ queues (multiplied by $\lambda_1\alpha$).

5.6 Step III: one-step policy improvement

5.6.1 Obtaining the improved policy

In the previous section we approximated $V(i,j,N)$ by $\widetilde{V}^{\alpha}(i,j)$. Next, we apply a technique called *one-step policy improvement*, introduced in [118], by inspecting the minimization term in Eq. (5.2), with $V(i,j,N)$ replaced by $\widetilde{V}^{\alpha}(i,j)$:

$$\min\left\{\gamma_1(i+1)+\gamma_2 j+\gamma_3+\widetilde{V}^{\alpha}(i+1,j);(N-T)^+ +\widetilde{V}^{\alpha}(i,j)\right\}. \quad (5.24)$$

Hence, the improved policy assigns a query to the DB if

$$\gamma_1(i+1)+\gamma_2 j+\gamma_3+\widetilde{V}^{\alpha}(i+1,j)\geq (N-T)^+ +\widetilde{V}^{\alpha}(i,j).$$

Substituting Eq. (5.22) and simplifying yields

$$\frac{\gamma_1}{1-\rho_1\alpha}(i+1)+\gamma_2 j+\gamma_3\geq (N-T)^+. \quad (5.25)$$

Note that this improved policy is independent of the constant C, as mentioned at the beginning of Section 5.5. Also, in the derivation of Eq. (5.25) we see that by choosing $\gamma_1(i+1)$ rather than $\gamma_1 i$ in the cost function, we obtain an expression where the α only occurs in front of the $(i+1)$ term. This allows us to intuitively explain the role of α: it acts as a tuning parameter of the improved policy, determining the influence of the number of queries i in the system on the decisions. For $\alpha=0$ the improved policy is independent of λ_1, but as α gets closer to 1 the number of queries in the system is weighed more heavily in the decision, and the policy becomes more biased towards the DB.

5.6.2 Determining α

The improved policy in Eq. (5.25) specifies a **class** of policies – only after choosing α (originally the parameter of the Bernoulli policy) do we have a concrete policy for which we can, e.g., determine average costs. However, we have no analytical relationship between $V(i,j,N)$ and $\widetilde{V}^{\alpha}(i,j)$, and thus determining α analytically is not possible. The best analytical option we have is to minimize $\tilde{g}^{\alpha}$ (of the Bernoulli policy applied to the simplified MDP) w.r.t. α, and use the resulting minimum for the improved policy. Unfortunately, subsequent experiments with value iteration demonstrate unsatisfactory performance of

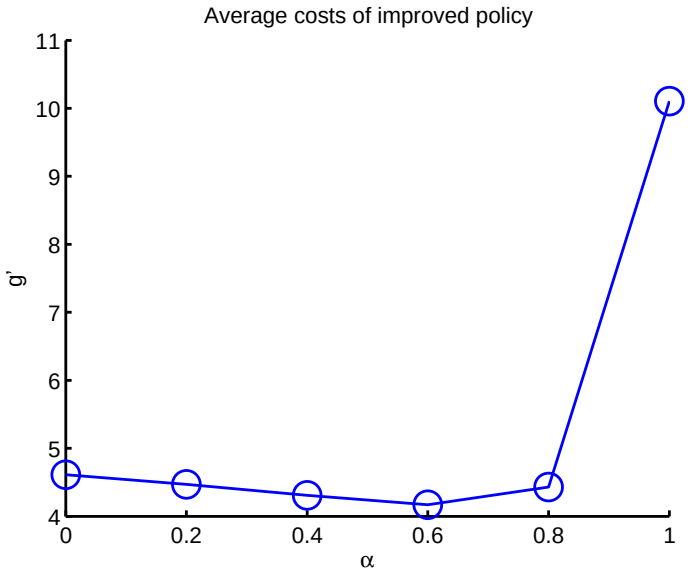


FIGURE 5.3: Average costs g' of the improved policy, for various values of α. The points in the graph are obtained with value iteration, using parameters $\mu_1 = \mu_2 = 0.3, T = 2, \gamma_1 = \gamma_2 = \gamma_3 = 3, \rho_1 = 0.8, \rho_2 = 0.1$. Our fitting approach for determining the minimum $\hat{\alpha}$ yields $\hat{\alpha} = 0.48$.

the resulting improved policy. We observed this behavior for various values for $\lambda_1, \lambda_2, \mu_1, \mu_2$, and T, so the unsatisfactory performance was general. The $\widetilde{V}^{\alpha}(i,j)$ and $\tilde{g}^{\alpha}$ do not approximate $V(i,j,N)$ and g from Eq. (5.2) sufficiently well.

Fortunately, a simple numerical approach allows us to compute an α that yields an improved policy with the desired near-optimal performance. To illustrate this procedure, consider Figure 5.3 showing approximations of the average costs g' of the improved policy (obtained with value iteration) as a function of α. The shape resembles a second-degree polynomial, and by carefully fitting such a polynomial to the approximate values, we can approximate $g'(\alpha)$. Then, we use the minimum $\hat{\alpha}$ of the fitted polynomial as input for the improved policy. Note that, due to this procedure, the improved policy is not an analytical policy: each time an improved policy is required, $\hat{\alpha}$ must be computed using the fitting procedure.

This approach for determining $\hat{\alpha}$ requires several approximate values α_i that together capture the shape of $g'(\alpha)$. They should be positioned such that the

minimum of the polynomial and that of $g'(\alpha)$ are at approximately the same α-value. Strictly speaking we need only three α-values to fit a second-degree polynomial. However, $g'(\alpha)$ is not truly a second-degree polynomial, and using four values results in a more appropriate fit in cases where $g'(\alpha)$ resembles the polynomial shape less. So how should we position these four points? In the next section we argue that the most interesting scenario from a practical point of view is one where ρ_1 is large. In this scenario, the number of queries i in the system is typically large. Recall that $\hat{\alpha}$ influences the improved policy in Eq. (5.25) via i: as $\hat{\alpha}$ gets closer to 1 the number of queries in the system is weighed more heavily in the decision, and the policy becomes more biased towards the DB. Hence, we should concentrate the fit of the polynomial on the right side of the interval, near $\alpha = 1$. Following this reasoning, we take $\alpha_1 = 0.25, \alpha_2 = 0.6, \alpha_3 = 0.85$, and $\alpha_4 = 0.95$.

The value of each $g'(\alpha_i)$ is obtained by running value iteration. The time needed to execute these four runs of value iteration should be shorter than the time needed to compute the optimal policy, otherwise there is no reason to use the improved policy. To this end, we do value iteration for the $g'(\alpha_i)$ on a much smaller state space than the one used for finding the optimal policy. Suppose that we run value iteration for the optimal policy on the truncated state space $\bar{\mathcal{X}} = [0, K_1] \times [0, K_2] \times [0, K_3]$ (in Section 5.7 we determine K_1, K_2, and K_3 experimentally in such a way that we avoid boundary effects). For the $g'(\alpha_i)$, we use the further truncated state space $\hat{\mathcal{X}} := \left[0, \lfloor \frac{K_1}{4} \rfloor\right] \times \left[0, \lfloor \frac{K_2}{4} \rfloor\right] \times \left[0, \lfloor \frac{K_3}{4} \rfloor\right]$. This effectively reduces the time needed to calculate $\hat{\alpha}$ (and thus also the improved policy) to a mere fraction of the time needed to obtain an optimal policy. The number by which K_1, K_2, and K_3 are divided (4) is determined experimentally to yield both low time-average costs and a short run time for the improved policy. Note that the further reduction of the state space is appropriate, because we do not require numerically accurate approximations of $g'(\alpha_1), \ldots, g'(\alpha_4)$. We only need to capture the general shape illustrated in Figure 5.3.

The complete procedure is as follows:

1. Calculate the bounds of the further truncated state space $\hat{\mathcal{X}}$.
2. For each of the values α_i, evaluate the improved policy using $\hat{\mathcal{X}}$ as state space, and record $g'(\alpha_i)$.
3. Fit a second-degree polynomial through $g'(\alpha_1), \ldots, g'(\alpha_4)$ using least squares.
4. Calculate the minimum of this polynomial, and use the α-value for which this minimum is attained as $\hat{\alpha}$.

In the example in Figure 5.3 this procedure yields $\hat{\alpha} = 0.48$, which agrees well with what the figure suggests. Figure 5.3 is generated with parameters $\mu_1 = \mu_2 = 0.3, T = 2, \gamma_1 = \gamma_2 = \gamma_3 = 3, \rho_1 = 0.8, \rho_2 = 0.1$, i.e., values corresponding to a high load on Q_1 and low load on Q_2. We expect a significant fraction of the queries to be assigned to Q_1, since a low load on Q_2 results in large N and thus using the DB is expensive. This observation is supported by the value $\hat{\alpha} = 0.48$ that our procedure yields for the improved policy. Also, the figure indicates that the sensitivity of the average costs $g'(\alpha)$ to α is minor around the minimum $\hat{\alpha}$.

5.7 Numerical results

In this section we experimentally inspect the performance of the improved policy by numerically comparing it to the optimal policy. Additionally, we compare a traditional age-threshold policy and a myopic policy to the optimal policy, allowing us to assess how the improved policy performs in relation to these other two policies. The three policies that we compare to the optimal policy listed in Eq. (5.26). We see that the age-threshold policy π^t ignores the load on the queueing system, and bases its actions solely on the age N. The myopic policy π^m takes the load of the system into account, by assigning queries to the DB or Q_1 based on the cost function in Eq. (5.1) only, ignoring the relative value function $V(i, j, N)$. In contrast, the improved policy π' is based on an approximation of the relative value function, and thus does include expectations about future query arrivals and report requests in its decisions. These expectations are captured by the parameter $\hat{\alpha}$, which determines how much emphasis the improved policy puts on the number of queries i in the system. Note that for $\hat{\alpha} = 0$ the improved and myopic policy are identical.

$$
\begin{aligned}
\pi^t(i,j,N) &= \begin{cases} \text{DB}, & \text{if } N \leq T, \\ Q_1, & \text{otherwise}, \end{cases} \\
\pi^m(i,j,N) &= \begin{cases} \text{DB}, & \text{if } \gamma_1(i+1) + \gamma_2 j + \gamma_3 \geq (N-T)^+, \\ Q_1, & \text{otherwise}, \end{cases} \\
\pi'(i,j,N) &= \begin{cases} \text{DB}, & \text{if } \frac{\gamma_1}{1-\rho_1\hat{\alpha}}(i+1) + \gamma_2 j + \gamma_3 \geq (N-T)^+, \\ Q_1, & \text{otherwise}. \end{cases}
\end{aligned}
\tag{5.26}
$$

Looking at our scenario, we expect that as $\rho_2 \to 1$, performance should be quite good, since the DB is refreshed often and thus most queries can be answered from the DB. Additionally, in situations with small ρ_1 the controller has to deal with only a small number of queries, costs are typically low, and the policies should demonstrate good performance. Hence, the most interesting part of the parameter space is where ρ_1 is high and ρ_2 is low (we call this the *critical region*). We structure our numerical analysis accordingly, by first inspecting the performance of the policies for $0 < \rho_1 \leq 0.8, 0 < \rho_2 < 1$, followed by an inspection of the critical region $0.7 < \rho_1 \leq 1, 0 < \rho_2 < 0.2$.

All numerical experiments below are done using the value iteration algorithm [159], and thus require a truncation of the state space $\mathcal{X} = \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0$ to $\bar{\mathcal{X}} = [0, K_1] \times [0, K_2] \times [0, K_3]$. Choosing the K_i must be done carefully to avoid the influence of boundary effects on the average costs. Tests on the three policies above, and on the optimal policy, suggests that a truncation to $\bar{\mathcal{X}} = [0, 200] \times [0, 200] \times [0, 200]$ is sufficient for $0 < \rho_1 \leq 0.8, 0 < \rho_2 < 1$. Increasing ρ_1 beyond 0.8 quickly adds boundary effects and requires a larger truncated state space: $\bar{\mathcal{X}} = [0, 300] \times [0, 300] \times [0, 300]$. Also, for value iteration we set the convergence criterion such that the procedure stops when the difference of the spans of two consecutive approximations is smaller than 0.001. Finally, we choose the parameters of the cost function in Eq. (5.1). We set $T = 2, \gamma_1 = \gamma_2 = \gamma_3 = 3$ and keep these fixed during all experiments.

In the following sections we numerically investigate the performance of our improved policy. First, we compare the three policies listed above to the optimal policy in Sections 5.7.1 (for the non-critical region) and 5.7.2 (for the critical region). Then in Section 5.7.3 we look at the time needed to calculate $\hat{\alpha}$, and thus the improved policy. Section 5.7.4 introduces a special random policy, where the controller flips a (fair) coin to decide which of the two actions to take. A large number of such policies are then compared to the three policies described above. Finally, in Section 5.7.5 we take a closer look at the optimal policy and its structure.

5.7.1 Analysis of region $0 < \rho_1 \leq 0.8, 0 < \rho_2 < 1$

In Figures 5.4 – 5.6 we inspect the performance of the three policies as compared to the optimal policy. We fix $\mu_1 = \mu_2 = 0.3$ and vary ρ_1 and ρ_2. The figures contain the difference in average costs with the optimal policy (in %), where the load ρ_2 on Q_2 is varied on the horizontal axis, and the load ρ_1 of Q_1 is reflected by the various lines. Figure 5.4 demonstrates that the simple

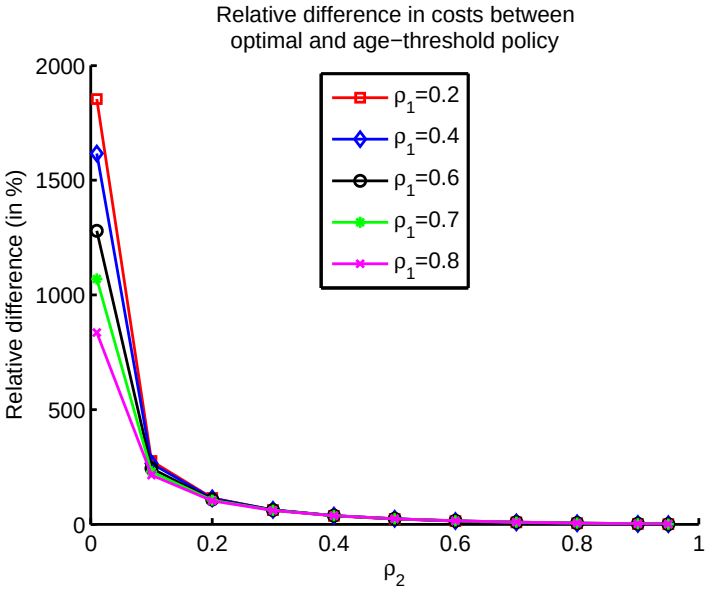


FIGURE 5.4: Relative difference in average costs of the age-threshold policy compared to the optimal policy.

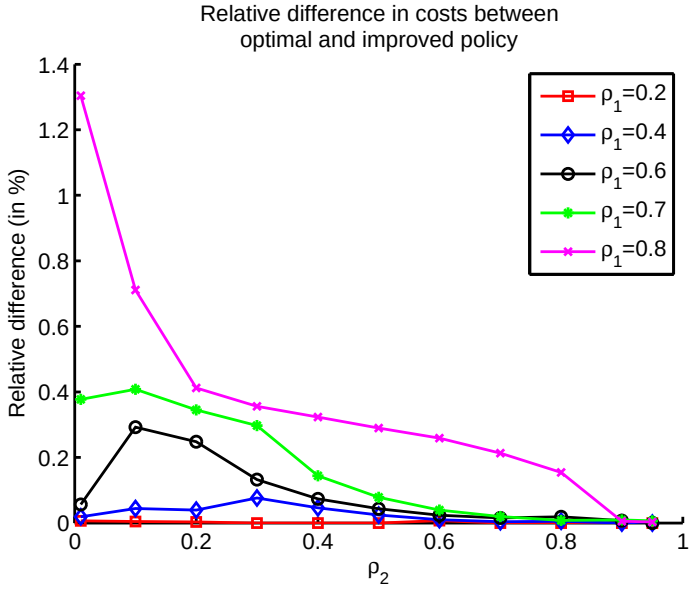


FIGURE 5.5: Relative difference in average costs of the improved policy compared to the optimal policy.

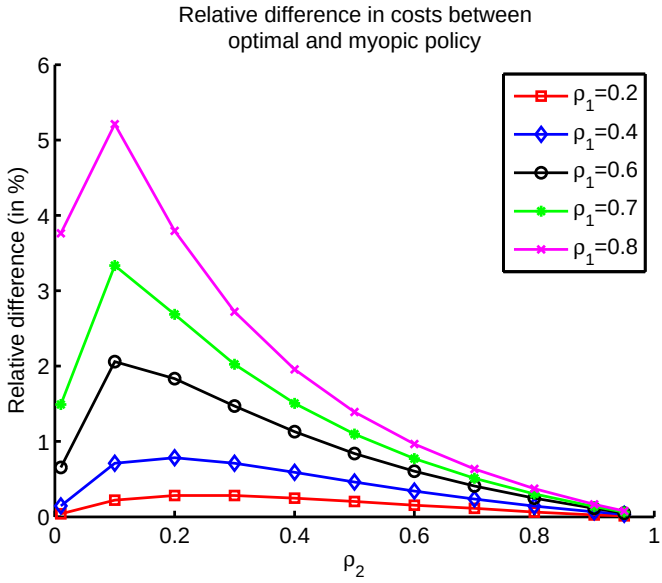


FIGURE 5.6: Relative difference in average costs of the myopic policy compared to the optimal policy.

age-threshold policy differs from optimality by as much as 2,000%. In contrast, the improved and myopic policies in Figures 5.5 and 5.6 are able to stay within 1.3% and 5.5% of optimality, respectively. Clearly, these policies perform significantly better than the age-threshold policy, so including the load of the queue system in the decision by the controller certainly is beneficial. Further inspection of Figures 5.4 – 5.6 reveals that the performance of the three policies degrades when ρ_1 and ρ_2 reach the critical region. We take a detailed look at this region in the next section.

5.7.2 Analysis of the critical region $0.7 < \rho_1 \leq 1, 0 < \rho_2 < 0.2$

We continue with a closer look at the critical region, i.e., the left-hand side of Figures 5.4 – 5.6, by repeating the corresponding numerical experiments for different values of ρ_1 and ρ_2 (again with $\mu_1 = \mu_2 = 0.3$) . The results are in Figures 5.7 – 5.9. As in the previous section, performance of the age-threshold policy is quite bad, with differences of up to 1,500%. Comparing Figures 5.8 to 5.9 clearly demonstrate that the improved policy has better overall performance than the myopic policy, with differences from optimality of at most 7% and 17%, respectively. The benefits of including the approximation to the relative value function in the improved policy are evident here.

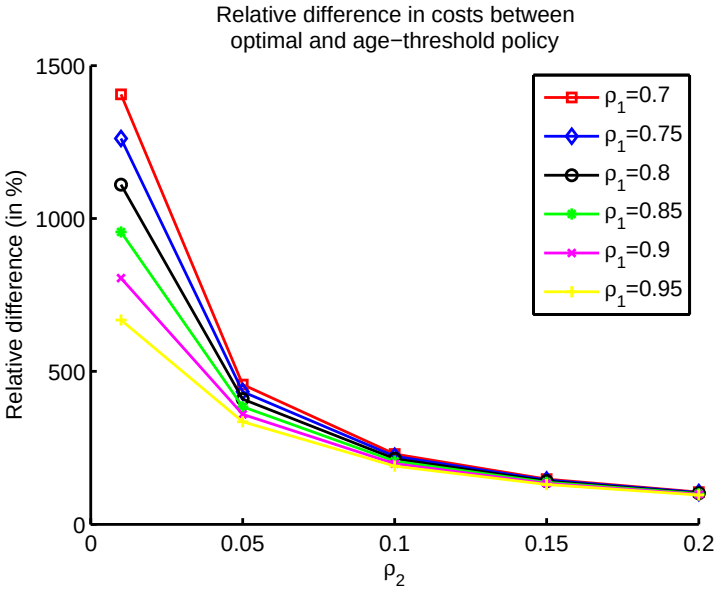


FIGURE 5.7: Again, the relative difference in average costs of the age-threshold policy compared to the optimal policy, but now inside the critical region.

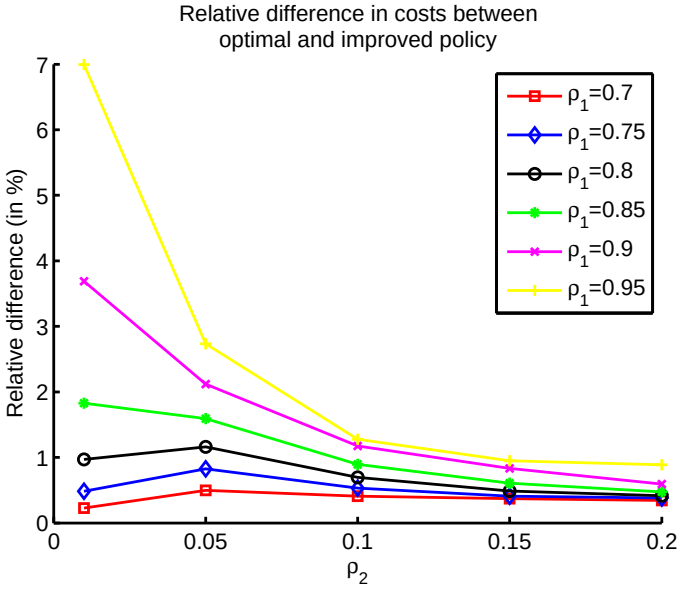


FIGURE 5.8: Again, the relative difference in average costs of the improved policy compared to the optimal policy, but now inside the critical region.

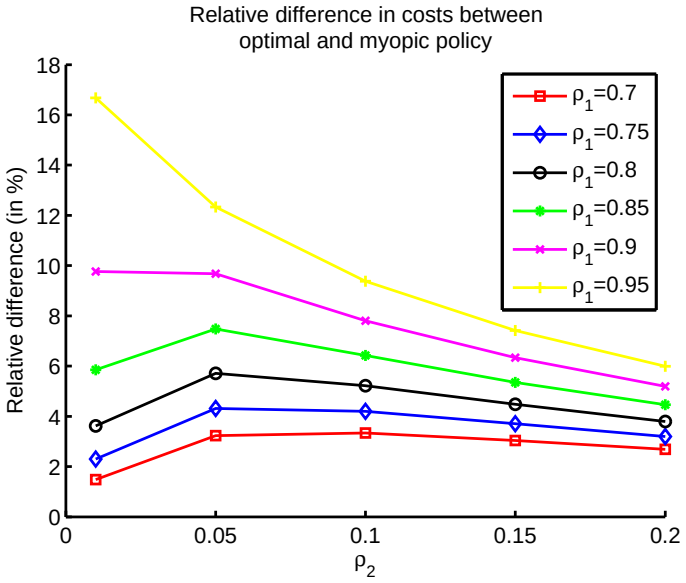


FIGURE 5.9: Again, the relative difference in average costs of the myopic policy compared to the optimal policy, but now inside the critical region.

Finally, Figures 5.8 and 5.9 demonstrate that the relative differences are not monotone. The left-most points (at $\rho_2 = 0.01$) seem to be closer to optimality than the points at $\rho_2 = 0.05$. Further experiments suggest that this is not caused by boundary effects. Also, the differences cannot be explained by the stopping criterion of value iteration, because the differences are too large. Since the observed feature is present in both figures, it seems likely that the optimal policy causes it, and thus that this behavior is a feature of the system. We return to this topic later in Section 5.7.5 when we talk about the optimal policy.

5.7.3 Computational complexity

As described in Section 5.6.2, the improved policy requires four short runs of the value iteration algorithm to determine the parameter $\hat{\alpha}$. The total duration of these runs should be less than the time required to find the optimal policy. Table 5.1 shows the time needed to find $\hat{\alpha}$ for the improved policy, divided by the time required to determine the optimal policy. As parameter values we use the same scenario as in Section 5.7.2, i.e., $\mu_1 = \mu_2 = 0.3$. The two tables clearly demonstrate that determining the improved policy is much faster than finding the optimal policy.

$\rho_1 \rightarrow$	**0.7**	**0.75**	**0.8**	**0.85**	**0.9**	**0.95**
$\rho_2 = 0.01$	0.0122	0.0061	0.0065	0.0059	0.0054	0.0054
$\rho_2 = 0.05$	0.0048	0.0069	0.0068	0.0076	0.0070	0.0069
$\rho_2 = 0.10$	0.0060	0.0058	0.0070	0.0067	0.0078	0.0077
$\rho_2 = 0.15$	0.0056	0.0054	0.0067	0.0063	0.0075	0.0061
$\rho_2 = 0.20$	0.0064	0.0063	0.0061	0.0071	0.0070	0.0068

TABLE 5.1: The run time for determining $\hat{\alpha}$ for the improved policy divided by the run time needed to obtain the optimal policy for various values of ρ_1 (columns) and ρ_2 (rows).

5.7.4 Model complexity

To get a feel for the complexity of the model in Eq. (5.2), we plot a so-called *Ordered Performance Curve* (OPC) [69]. Each point in this plot shows the average costs of a policy that we generate randomly: at each state (i, j, N) we choose action $a = \{Q_1\}$ with probability 0.5, or $a = \{DB\}$ otherwise. By repeating this procedure, we create 2,500 such policies, evaluate them, and plot their average costs in Figure 5.10. Additionally, this figure shows the average costs of the optimal policy and (in our case) of the improved, the age-threshold, and myopic policies. The parameters are $\mu_1 = \mu_2 = 0.3, \rho_1 = 0.8, \rho_2 = 0.1$, based on the critical region in the parameter space. The random policies all perform better than the age-threshold policy, and once again confirm that there is room for improvement on such a traditional policy.

Since the markers of the optimal, improved, and myopic policies are indistinguishable in Figure 5.10, the fifteen best policies are plotted again in Figure 5.11. The steep slope on the left of both figures illustrates that none of the randomly selected policies is able to closely match the performance of the optimal policy. Hence, the plot demonstrates that the performance of the improved policy is not easily replicated by a random policy, and that including the load of the system in the decision policy is meaningful.

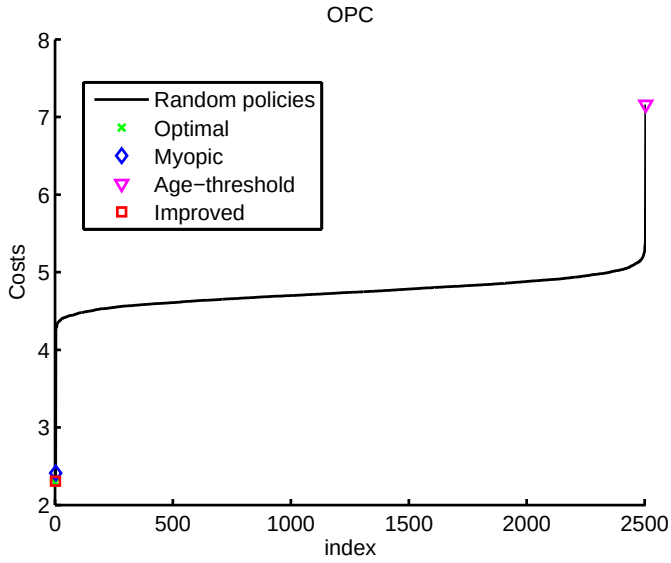


FIGURE 5.10: Ordered Performance Curve - costs of 2,500 randomly selected policies, as well as the optimal, improved, myopic, and age-threshold policies. The age-threshold policy clearly performs badly.

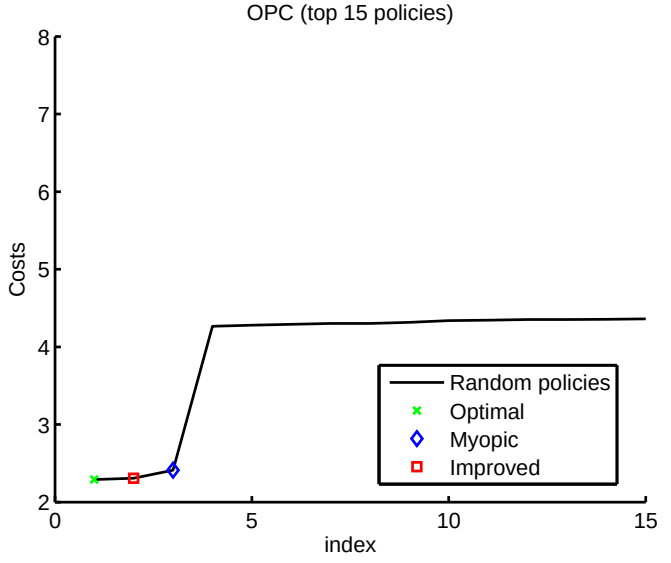


FIGURE 5.11: The same OPC as in Figure 5.10, but now only for the fifteen best policies. The improved and myopic policies are both close to the optimal policy, and perform significantly better than the best random policy.

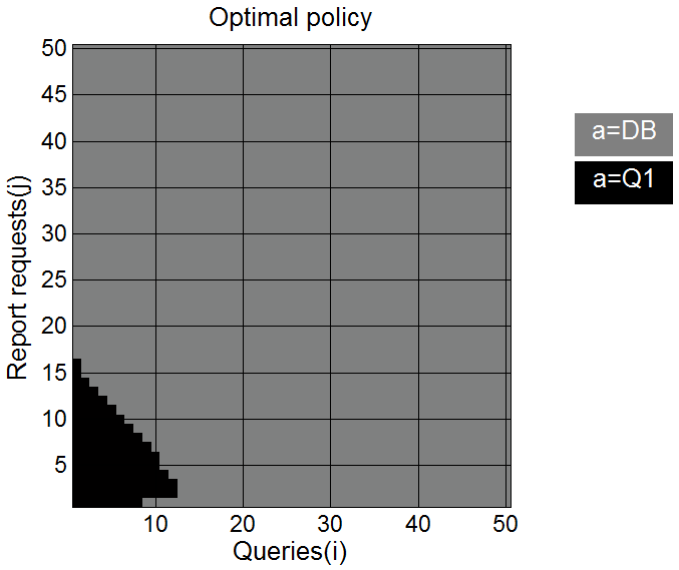


FIGURE 5.12: The optimal policy for $N = 55$, with gray indicating that action $a = \text{DB}$ is taken, and black that $a = Q_1$ is taken.

5.7.5 The optimal policy

Next, we inspect the optimal policy in Figures 5.12 and 5.13, again using parameters $\mu_1 = \mu_2 = 0.3, \rho_1 = 0.8, \rho_2 = 0.1$ from the critical region. The first shows a cross-section of the optimal policy at $N = 55$, the second at $N = 120$. Here, for each grid point (i, j) the color gray indicates that action $a = \text{DB}$ is taken and black that $a = Q_1$. The figures suggest that (away from the boundaries) the optimal policy is a hyperplane in three-dimensional space, i.e., a switching policy. This observation is supported by intuitions about the problem scenario: once Q_1 reaches a certain load, the controller switches to using the DB, and continue to do so as the load increases. Hence, an optimal policy with a switching structure is in line with our expectations. We were unable to verify this structure mathematically, but we expect that a proof is feasible. The conjecture below formalizes the claim:

CONJECTURE 5.7.1 (Asymptotic switching policy). *The optimal policy for the* MDP *in Eq. (5.2) is a switching curve for N sufficiently large.*

Looking at Figures 5.12 and 5.13, we see that the optimal policy is cropped near the boundary $\{j = 0\}$ of the state space. This effect is caused by the interaction between the number of report requests j and the costs $(N-T)^+$ for

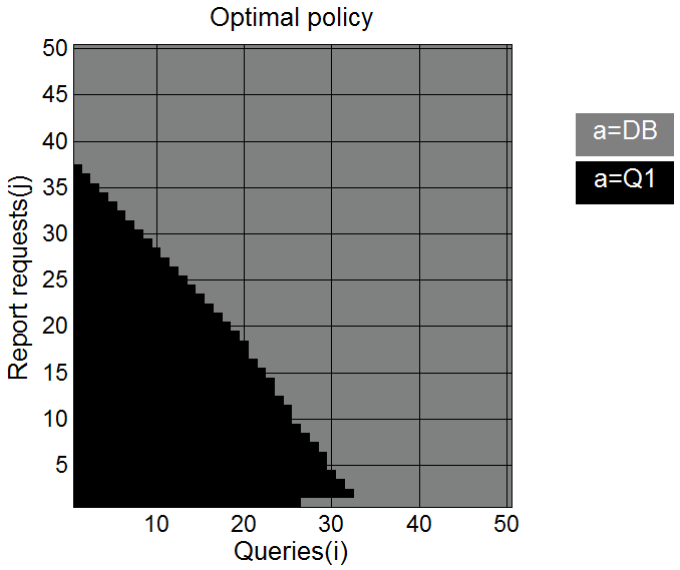


FIGURE 5.13: The optimal policy for $N = 120$, with gray indicating that action $a = \text{DB}$ is taken, and black that $a = Q_1$ is taken.

DB assignments. They are connected via N using the term $\mu_2 V(i, j-1, 0)\mathbb{1}_{\{j>0\}}$ in Eq. (5.2), which drops out at the boundary $\{j = 0\}$ of the state space. Consequently, on the boundary the connection between j and N is severed, and changes the structure of the MDP and the optimal policy significantly. This also explains the observation in Section 5.7.2 that the performance of the improved and myopic policies changes for $\rho_2 \approx 0$.

Still, in situations where the boundary $\{j = 0\}$ of the state space is not reached frequently, we expect switching policies to perform well since the boundary effect is relatively small. This is supported by the results on our improved policy and the myopic policy (both are switching policies) in the previous sections.

5.8 Conclusion

In this chapter we investigated the trade-off between data freshness and query response times. We formulated this trade-off as a Markov Decision Process with a three-dimensional state space. The resulting model contained several complication aspects, preventing a derivation of an analytical expression for the

optimal policy. Instead, we introduced a three-step approach to finding an approximate policy with near-optimal performance. The first step demonstrated how the original three-dimensional model can be approximated by a simpler two-dimensional model that still captures the important dynamics. Then, in the second step, we described how this simpler model can be solved analytically, using differencing techniques to deal with the inhomogeneous terms. In step three we applied one-step policy improvement to construct our approximate policy. Finally, we numerically demonstrated that this improved policy has near-optimal performance, and significantly outperforms both an age-threshold policy and a myopic policy. Moreover, the experiments reveal that there is room for improvement on the traditional age-threshold policy, which is commonly used in practice. Future research directions include making the improved policy analytic by removing the dependence on short runs of value iteration for determining the parameter α (see also the remark at the end of Section 6.7), and proving the conjecture that the optimal policy is asymptotically a switching curve.

5.9 Appendix: uniqueness

In Section 5.5 we solved the two-dimensional difference equation (5.6), known in MDP literature as the *Poisson equation*. For this equation we have only one boundary condition $V(0,0) = 0$, which is not enough to completely determine the solution. Consequently, after solving the difference equation the constant $\widetilde{V}_1^{\alpha}(0)$ is yet to be determined in Eq. (5.21).

In order to investigate uniqueness we repeat arguments from Chapter 2 and 4 of [27]. First, note that Eq. (5.6) induces a *Markov cost chain* with transition matrix P, state space $\mathcal{X} = \mathbb{N}_0 \times \mathbb{N}_0$, and cost function

$$c(i,j) = \lambda_1 \alpha \left[\gamma_1(i+1) + \gamma_2 j + \gamma_3\right] + \lambda_1(1-\alpha)C.$$

Denote with $\mathbb{B}(\mathcal{X})$ the Banach space of bounded real-valued functions u on $\mathcal{X}$ with the supremum norm, i.e., the norm $||\cdot||$ defined by

$$||u|| = \sup_{(i,j)\in\mathcal{X}} |u(i,j)|\,.$$

Conventional uniqueness proofs for Markov cost chains rely on bounded cost functions contained in $\mathbb{B}(\mathcal{X})$. However, our cost function $c(i,j)$ is unbounded and thus not contained in $\mathbb{B}(\mathcal{X})$. A remedy to this situation is to consider suitable larger Banach spaces instead of $\mathbb{B}(\mathcal{X})$. In order to construct such a

space, consider a *weight function* $w : \mathcal{X} \to [1, \infty)$. The *w-norm* is then defined by

$$||u||_w = \sup_{(i,j)\in\mathcal{X}} \frac{|u(i,j)|}{w(i,j)}.$$

A function u is said to be w-bounded if $||u||_w < \infty$, and the space of all w-bounded functions is denoted by $\mathbb{B}_w(\mathcal{X})$. We also define the matrix norm related to $||\cdot||_w$ as $||A||_w = \sup\{||Au||_w : ||u||_w \leq 1\}$. This norm can be rewritten in the following equivalent form (see Eq. (7.2.8) in [67])

$$||A||_w = \sup_{x\in\mathcal{X}} \sum_{y\in\mathcal{X}} \frac{|A_{xy}|\, w(y)}{w(x)}.$$

Finally, we introduce the taboo transition matrix ${}_M P$ as

$${}_M P_{xy} = \begin{cases} P_{xy}, & y \neq M, \\ 0, & y \in M, \end{cases}$$

with $x, y \in \mathcal{X}$ and in our case $M = (0, 0)$. We now state a property and adapted theorem from [27] on uniqueness of solutions of Eq. (5.6).

Property 5.9.1 (page 19 of [27]). *A Markov chain is called w-geometrically recurrent with respect to M [w-GR(M)] if there exists an $\epsilon > 0$ such that $||{}_M P||_w \leq 1 - \epsilon$.*

Theorem 5.9.2 (Lemma 2.1 combined with Theorem 2.10 of [27]). *Suppose that the Markov chain induced by a policy π is unichain, stable, aperiodic, and w-GR(M). Let both (g, V) and (g', V') be solutions to the Poisson equation. Then $g = g'$ and the relative value functions V and V' differ by only a constant.*

In our case, the Bernoulli policy does indeed induce a Markov chain that is unichain, stable, and aperiodic. The key to ensuring uniqueness is choosing a suitable weight function w such that Property 5.9.1 is satisfied. Section 3.4 of [146] shows that a suitable weight function is of the form

$$w(i,j) = K \prod_{k=1}^{i} (1 + m_k) \prod_{l=1}^{j} (1 + n_l),$$

where $\{m_k\}$, $\{m_l\}$, and K are constants. Unfortunately, the expressions involved are cumbersome and not easy to state explicitly, making it difficult for us to illustrate the construction of the weight function. In the remainder of this section we make an additional assumption that allows us to find a weight

function that is explicit. This assumption is only made to facilitate explicitness, and readers interested in the case without the assumption are referred to [146].

Following Section 4.1 of [27], we assume that $\rho_1\alpha + \rho_2 < 1$. The non-zero entries in the transition matrix are given by

$$\begin{aligned}
P_{(i,j)(i+1,j)} &= \lambda_1\alpha,\\
P_{(i,j)(i,j+1)} &= \lambda_2,\\
P_{(i,j)(i-1,j)} &= \mu_1\mathbb{1}_{\{i>0\}},\\
P_{(i,j)(i,j-1)} &= \mu_2\mathbb{1}_{\{j>0\}},\\
P_{(i,j)(i,j)} &= 1 - P_{(i,j)(i+1,j)} - P_{(i,j)(i,j+1)} - P_{(i,j)(i-1,j)} - P_{(i,j)(i,j-1)}.
\end{aligned}$$

Set $w(i,j) = (1+k_1)^i(1+k_2)^j$ for some constants k_1 and k_2. Now consider

$$||_M P||_w = \sum_{(i',j')\neq(0,0)} \frac{P_{(i,j)(i',j')}w(i',j')}{w(i,j)},$$

which is given by

$$\begin{cases}
\lambda_1\alpha(1+k_1) + \lambda_2(1+k_2), & (i,j) = (0,0),\\
\lambda_1\alpha k_1 + \lambda_2 k_2 + 1 - \mu_1, & (i,j) = (1,0),\\
\lambda_1\alpha k_1 + \lambda_2 k_2 + 1 - \mu_2, & (i,j) = (0,1),\\
\lambda_1\alpha k_1 + \lambda_2 k_2 + 1 - \dfrac{\mu_1 k_1}{1+k_1}, & i > 1, j = 0,\\
\lambda_1\alpha k_1 + \lambda_2 k_2 + 1 - \dfrac{\mu_2 k_2}{1+k_2}, & i = 0, j > 1,\\
\lambda_1\alpha k_1 + \lambda_2 k_2 + 1 - \dfrac{\mu_1 k_1}{1+k_1} - \dfrac{\mu_2 k_2}{1+k_2}, & i > 0, j > 0.
\end{cases}$$

We need to choose k_1 and k_2 such that all expressions are strictly less than 1. Observe that if the fourth and fifth expression are less than 1, then all others are also satisfied. Hence, we can restrict our attention to the system

$$\begin{aligned}
f_1(k_1,k_2) &= 1 + \lambda_1\alpha k_1 + \lambda_2 k_2 - \frac{\mu_1 k_1}{1+k_1},\\
f_2(k_1,k_2) &= 1 + \lambda_1\alpha k_1 + \lambda_2 k_2 - \frac{\mu_2 k_2}{1+k_2},
\end{aligned}$$

with the assumptions $\lambda_1\alpha + \lambda_2 + \mu_1 + \mu_2 < 1$ and $\rho_1 + \rho_2 < 1$.

Observe that $f_1(0,0) = f_1\big((\mu_1 - \lambda_1\alpha)/(\lambda_1\alpha), 0\big) = 1$. Thus, the points $(0,0)$ and $\big((\mu_1 - \lambda_1\alpha)/(\lambda_1\alpha), 0\big)$ lie on the curve $f_1(k_1,k_2) = 1$. Furthermore, k_2

satisfies $k_2 = \mu_1/\lambda_2 - \mu_1/\big(\lambda_2(1+k_1)\big) - \lambda_1\alpha/\lambda_2$. Note that this function has a maximum value at $k_1 = \sqrt{\mu_1/(\lambda_1\alpha)} - 1$. Hence, this description determines the form of f_1; the curve $f_1(k_1, k_2) = 1$ starts in $(0,0)$ and increases to an extreme point, and then decreases to the k_1-axis again. The curve f_2 has a similar form, but with the role of the k_1-axis interchanged with the k_2-axis.

The curves determine an area of points (k_1, k_2) such that f_1 and f_2 are strictly less than one if the partial derivative to k_1 at $(0,0)$ of the curve $f_1(k_1, k_2) = 1$ is greater than the partial derivative to k_2 of the curve $f_2(k_1, k_2) = 1$ at $(0,0)$. These partial derivatives are given by $(\mu_1 - \lambda_1\alpha)/\lambda_2$ and $\lambda_1\alpha/(\mu_2 - \lambda_2)$, respectively. Since $\rho_1\alpha + \rho_2 < 1$, we have $\lambda_1\alpha\mu_2 + \lambda_2\mu_1 < \mu_1\mu_2$. Adding $\lambda_1\alpha\lambda_2$ to both sides gives $\lambda_1\alpha\lambda_2 < \mu_1\mu_2 - \lambda_1\alpha\mu_2 - \lambda_2\mu_1 + \lambda_1\alpha\lambda_2 = (\mu_1 - \lambda_1\alpha)(\mu_2 - \lambda_2)$. Hence, the relation $\lambda_1\alpha/(\mu_2 - \lambda_2) < (\mu_1 - \lambda_1\alpha)/\lambda_2$ holds. Thus, indeed the partial derivative to k_1 at $(0,0)$ of the curve $f_1(k_1, k_2) = 1$ is greater than the partial derivative to k_2 of the curve $f_2(k_1, k_2) = 1$ at $(0,0)$, and there is an area of pairs (k_1, k_2) such that the Markov chain is w-GR(M). For these points it holds that $(1 + k_n) < 1/\rho_n$ for $n = 1, 2$. Observe that any sphere with radius $\epsilon > 0$ around $(0,0)$ has a non-empty intersection with this area. Hence, the cost function cannot contain terms in i and/or j that grow exponentially fast to infinity, and neither can the relative value function. Consequently, we need to choose $\widetilde{V}_1^{\alpha}(0)$ in Eq. (5.21) such that the exponential term $\big(\frac{\mu_1}{\lambda_1\alpha}\big)^i$ disappears.

6

Value Function Discovery in Markov Decision Processes with Evolutionary Algorithms

In this chapter we introduce a novel method for discovery of relative value functions for Markov Decision Processes (MDPs). This method, which we call Value Function Discovery (VFD), is based on ideas from the Evolutionary Algorithm field. VFD's key feature is that it discovers descriptions of relative value functions that are algebraic in nature. This feature is unique, because the descriptions include the model parameters of the MDP. The algebraic expression of the relative value function discovered by VFD can be used in several scenarios, e.g., conversion to a policy (with one-step policy improvement) or control of systems with time-varying parameters.

The work in this chapter is a first step towards exploring potential usage scenarios of discovered relative value functions. We give a detailed description of VFD and illustrate its application on an example MDP. For this MDP we let VFD discover an algebraic description of a relative value function that closely resembles the optimal relative value function. The discovered relative value function is then used to obtain a policy, which we compare numerically to the optimal policy of the MDP. The resulting policy has near-optimal performance on a wide range of model parameters. Finally, we identify and discuss future application scenarios of discovered relative value functions.

This chapter is based on the results presented in [5].

6.1 Introduction

When dealing with MDPs, various techniques are available to, e.g., obtain optimal policies for decision making. These techniques fall into two categories, namely numeric and algebraic techniques. In the former category, the most well-known methods are value iteration, policy evaluation, and policy iteration [130]. Value iteration is an iterative technique for finding an optimal control policy and the corresponding time-average cost. With policy evaluation one can find the time-average cost of a given policy, and policy iteration improves and evaluates policies iteratively. The aforementioned techniques are numeric in nature, so when, e.g., the model parameters change they have to be reapplied to the updated scenario. Ideally, one would like to solve an MDP algebraically and obtain the optimal policy (with the model parameters included). This approach is, however, often not feasible due to complexities of the model.

Motivated by this, we introduce a novel method called *Value Function Discovery* that is aimed at obtaining an algebraic description of a relative value function. In essence, VFD fits an algebraic function through several sample points of the relative value function. The fitting procedure is based on a technique from the Evolutionary Algorithm (EA) family known as Genetic Programming (GP). By including sample points for various model parameters, VFD can also include these parameters when discovering a relative value function. After applying VFD, the relative value function can be used to, e.g., obtain an algebraic policy. In the current chapter we use this to demonstrate that, for an example MDP, the relative value function discovered by VFD yields a near-optimal policy.

In the remainder of this chapter we describe VFD and illustrate it by applying VFD to an example MDP. We start with a review of related work in Section 6.2, and an introduction to GP in Section 6.3. Then, we continue with a detailed description of VFD in Section 6.4 and of the example MDP in Section 6.5. Numerical results are presented in Section 6.6, followed by a discussion in Section 6.7 and concluding remarks in Section 6.8.

6.2 Related work

The literature combining EAs and MDPs mostly uses EAs to learn policies (contrary to VFD, which learns relative value functions). In [35] the authors introduce evolutionary policy iteration, where the policy improvement step is

integrated with an EA to iteratively obtain better policies. This procedure is shown to have monotone convergence for finite action spaces. The authors of [71] enhance the work in [35] by generating policies in the population via sub-MDPs, thereby speeding up convergence. From an application perspective, [161] provides an example of how EAs and MDPs can be used in a practical scenario. [24] compares an EA to policy iteration, and provides a useful reminder that policy iteration typically converges quickly and thus often outperforms an EA-approach.

Closest to our research is [93] by Lin et al., where the authors construct a piecewise linear approximation of the relative value function. In this approach, the linear elements are learned using a Genetic Algorithm. Like VFD, Lin's approach results in an approximation of the relative value function. However, the relative value function discovered by VFD is a closed-form expression, whereas [93] finds a piecewise linear approximation. Having a closed-form expression is preferable when, e.g., studying the structure of the MDP using the discovered relative value function. Also, [93] focuses on convex relative value functions, and VFD does not make any assumptions about the structure of the relative value function. Another difference is the type of EA that is used: [93] employs a Genetic Algorithm, whereas VFD is based on GP. In particular, [93] does not use the tree-based representation inherent to GP. Finally, [93] does not allow for the placement of model parameters in the approximate relative value function.

A paper that does use GP in an MDP-context is [53]. The authors loosely explore the combination of GP and MDPs on an example of a war game and demonstrate that it performs well compared to a pure MDP-based technique. Their approach differs from the one described in this chapter, because they use GP to learn policies and not relative value functions, as VFD does.

Summarizing, the distinguishing feature of VFD is its focus on discovering relative value functions. Although existing methods in literature choose to learn policies, learning relative value functions has significant advantages as well. In particular, VFD has the following benefits:

- VFD applied to an optimal relative value function yields policies with near-optimal performance.
- For MDPs that allow for an explicit closed-form expression of the optimal relative value function, VFD can find this optimal relative value function with arbitrary precision. Thus, it can also find the optimal policy for such MDPs. We present an illustration of this in Section 6.6.6.

- VFD produces an algebraic expression of a policy that includes the parameters of the MDP. Hence, the policy is still applicable if the parameters of the model change in value. This allows for dynamic control in time-varying systems, without making the underlying model time-dependent.
- Relative value functions discovered by VFD can help gain an understanding of the structure of the optimal relative value function, policy, and model.
- Alternative techniques for analyzing MDPs often require knowledge of structural properties of the relative value function (e.g., gradient-based methods such as local search). These properties can be discovered by VFD.
- For many MDPs a near-optimal policy does not require an extremely accurate fit of the optimal relative value function. Thus, learning relative value functions can quickly result in good policies.
- VFD works with any MDP without requiring any changes to the algorithm.

6.3 Genetic programming

Since VFD is based on GP, we give a short description of this technique in this section. Readers interested in a more detailed treatment of GP are referred to books [44, 125, 142]. The general idea of GP is to maintain a population of individuals and iteratively attempt to improve this population over several generations. In each generation (i.e., an iteration step), the current population form new offspring by combining individuals. The main idea underlying GP is that combining good individuals leads, over time, to offspring that are better than their predecessors. Below we describe this procedure in more detail, with particular attention for the *mutation* and *recombination* operators used for generating offspring. Determining the quality of an individual is related to VFD's application of GP to MDPs, so we postpone it until Section 6.4.

In GP, each individual in the population is an algebraic expression represented by a tree, and later we use such trees to represent the relative value function of an MDP. Figure 6.1A illustrates a tree representation of the function $V(x) = \frac{x(x+1)}{2\mu(1-\rho)}$ (the relative value function of an $M/M/1$ queue [28]). The operators $\{/, *, +, -\}$ from this expression are in the internal nodes of the tree, whereas the leafs contain the variables (x), parameters (ρ, μ), and constants $(1, 2)$. In this chapter we only use the operators $\{/, *, +, -\}$ from the example, but the representation is flexible and also allows for, e.g., exponents, square roots, and logarithms. Also, note that a representation of a function by a

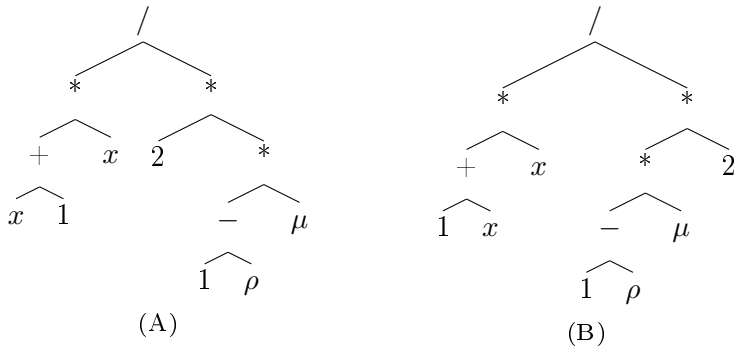


FIGURE 6.1: Two trees, each a representation of $V(x) = \frac{x(x+1)}{2\mu(1-\rho)}$.

tree is not unique: the tree in Figure 6.1B is also a valid representation of $V(x) = \frac{x(x+1)}{2\mu(1-\rho)}$. Unicity of representation is, however, not required by VFD. In fact, this feature is used by VFD to include a preference for short trees.

GP uses the mutation and recombination operator to generate new offspring from an existing population. In particular, the recombination operator generates two new offspring from two parents, and the mutation operator produces one new offspring from one parent. The recombination operator takes the following two steps:

1. Randomly select a node in each of the two trees.
2. Exchange the two subtrees.

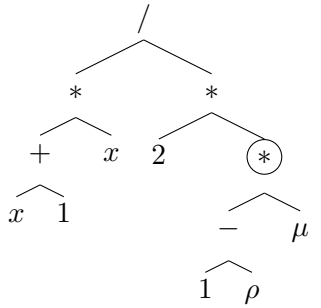
The procedure is illustrated in Figure 6.2, where recombination is applied to the two trees in Figures 6.2A and 6.2B. The subtree with the encircled $*$ as root in Figure 6.2A is exchanged with the subtree with root / (also encircled), resulting in the trees in Figures 6.2C and 6.2D. This combines the functions

$$V(x) = \frac{x(x+1)}{2\mu(1-\rho)} \quad \text{and} \quad V(x) = x\frac{1}{\mu} + x + 3.3,$$

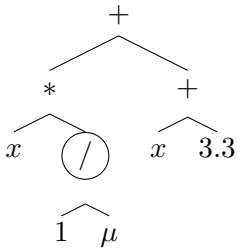
to, respectively,

$$V(x) = \frac{x(x+1)}{2\frac{1}{\mu}} \quad \text{and} \quad V(x) = x(1-\rho)\mu + x + 3.3,$$

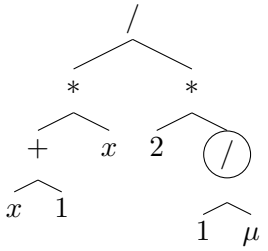
Mutation of trees is similar to recombination, except that a selected subtree is removed and replaced by a randomly generated subtree. The procedure is:



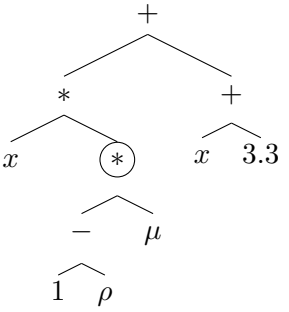
(A) Tree 1 before recombination.



(B) Tree 2 before recombination.



(C) Tree 1 after recombination.



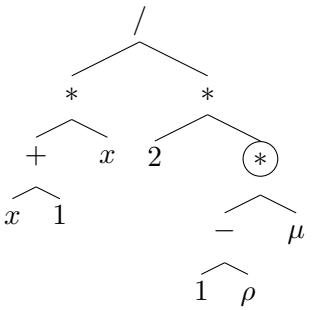
(D) Tree 2 after recombination.

FIGURE 6.2: The recombination operator illustrated on the two trees in Figures 6.2A and 6.2B. The encircled subtrees are exchanged, resulting in the trees in Figures 6.2C and 6.2D.

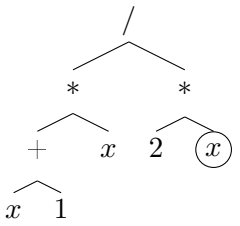
1. Select one of the nodes of the tree uniformly at random.
2. Remove this node and the subtree attached to it.
3. Randomly generate a new subtree.
4. Insert this new subtree in the place of the old subtree.

Figure 6.3 illustrates the procedure for the tree for $V(x) = \frac{x(x+1)}{2\mu(1-\rho)}$ which we saw earlier, displayed again in Figure 6.3A. The circled node is selected for mutation and removed from the tree, together with its subtree. It is replaced by a randomly generated subtree, in this case a simple tree with only one element (x). The result is shown in Figure 6.3B, with the newly added tree encircled. Thus, mutation changes $V(x)$ from $\frac{x(x+1)}{2\mu(1-\rho)}$ to $\frac{x(x+1)}{2x}$.

Applying the GP paradigm with only the recombination operator already results in the desired improvement of the population over time. This improvement is, however, limited by the information present in the population at the



(A) Before mutation.



(B) After mutation.

FIGURE 6.3: Mutation removes the subtree of the encircled node in Figure 6.3A (representing the term $\mu(1-\rho)$) and replaces it by a randomly generated subtree. The new subtree contains, in this case, only the element x and is encircled in Figure 6.3B.

start of the algorithm. The mutation operator is used by GP to insert new information into the population. The performance of GP is determined partly by carefully balancing the application of the mutation and recombination operators.

6.4 Value Function Discovery

VFD applies GP to relative value functions of an MDP. Each individual in the population corresponds to a potential relative value function, and by repeatedly modifying these trees with the mutation and recombination operators, VFD discovers new relative value functions. In order to judge the quality of a tree in the population, VFD compares it to several sample points of the actual relative value function. If a tree 'closely' matches the sample points, it is considered a 'good' relative value function (this is made precise in Section 6.4.7). By including sample points for various MDP parameter values, VFD is able to include these parameters in the tree representation, and thus in the discovered relative value function.

In the remainder of this section we describe VFD in more detail, with particular attention for the preparation of the sample points, several GP-specific aspects that were omitted from Section 6.3, and determining the quality of the fit of a tree to the sample points.

6.4.1 Preparing sample points

Before VFD starts, it requires input from the MDP in the form of sample point sets. The sample points in each set are generated by fixing the values of the model parameters, finding the relative value function numerically (with, e.g., value iteration), and then selecting appropriate sample points. By repeating these steps for multiple values of the model parameters, they can be included in VFD. The steps are described below, along with some supporting notation:

1. Fix values for each of the m model parameters.
2. Find a numeric approximation of the relative value function of the MDP (by, e.g., value iteration).
3. Select several sample points that together capture the shape of the relative value function. Each sample point is denoted by s, and the pairs $(s, V(s))$ together form the sample point set $\mathcal{S}_q$.
4. Save these sample points into a file.
5. Repeat steps 1–4 for several combinations of the m parameters. We denote the resulting number of sample point sets by Q, and each sample point set by $\mathcal{S}_q$, with $q \in [0, Q-1]$.

In these steps, several choices are made based on the MDP that VFD is applied to: (a) the values for the m parameters in each sample point set in step 1, (b) the selection of the sample points in each set in step 3, and (c) the number of sample point sets Q in step 5. Section 6.5.2 illustrates the considerations for making these choices on an example MDP.

Preparing the sample point sets requires running, e.g., value iteration on the MDP, which yields an optimal policy. So why not use this policy instead of running VFD? Well, the policy found by value iteration is numeric in nature, whereas VFD produces an algebraic policy. Consequently, the policy resulting from VFD can be applied to model parameters that are not used to generate the sample points. This feature is illustrated later in this chapter in Section 6.6.4, when we apply VFD to an example MDP.

In this chapter we use value iteration for generating the sample point sets, but for applying VFD other techniques can be used as well. For instance, when the MDP is too large for running value iteration, one can also use TD-learning [148], which provides numerical approximations of the relative value function using simulations.

Algorithm 6.1 Value function discovery (VFD)

```
 1: function VFD( )
 2:     samplePointSets ← readSamplePointSets()
 3:     population ← initPopulation()
 4:     while not isConverged() do
 5:         repeat
 6:             if apply mutation then
 7:                 children ← mutate(selectParent())
 8:             else
 9:                 children ←recombine(selectParent(),
10:                                     selectParent())
11:             end if
12:         until LAMBDA children generated
13:         setError(children)
14:         population ← population + children
15:         sort(population)
16:         survivorSelection()
17:         if not isPopulationDiverse() then
18:             initPopulation()
19:         end if
20:     end while
21:     return population[0]
22: end function
```

6.4.2 Overview

A pseudo code listing of VFD is shown in Algorithm 6.1, and in the following paragraphs we describe the steps involved. We start with a high-level description in Algorithm 6.1, and then move on to a detailed description of the functions involved (Algorithms 6.2 and 6.3). During these descriptions we encounter the first of several parameters of VFD, which are listed in Table 6.1 (together with assigned values that we use later in the example MDP in Section 6.5). Functions and parameters are written in SMALLCAPS throughout the text, including trailing brackets () for functions.

The algorithm starts at line 2 by loading the sample point sets of the MDP from the files. These are used later to determine the error of a tree. Next, the population is initialized by filling it with MU randomly generated trees. Lines 4–20 describe the steps taken by GP: first, LAMBDA children are generated using mutation and recombination (lines 6–11). Then, their error is calculated, they

are added to the population, and the population is sorted from smallest error to largest (lines 13-15). Survivor selection removes LAMBDA trees from the population, leaving MU individuals (line 16). This procedure is repeated until convergence (line 4).

Repeating the GP-like procedure described above eventually leads to a population where most trees are the same or similar. When this happens, the algorithm loses its ability to learn and evolve, and the population is said to have lost *diversity*. VFD deals with this by checking the level of diversity in each generation (with the ISPOPULATIONDIVERSE() function at line 17). When this check indicates that too much diversity has been lost, VFD reinitializes the population (line 18) with random trees and restarts the search process. Upon convergence VFD returns the discovered tree (line 21).

6.4.3 Mutation, recombination, diversity, and convergence

Next we describe the functions used in Algorithm 6.1 in more detail, starting with the MUTATE() function at line 1 of Algorithm 6.2. Mutation occurs according to the GP paradigm, as described in Section 6.3: a random point in the tree is selected (line 2) and the subtree at that point is replaced by a randomly generated subtree (lines 3 and 4). Similarly, the recombination operator is represented by the RECOMBINE() method. Both functions rely on a numbering of the nodes in a tree, which VFD assigns using a root-left-right walk of the tree.

Each time that VFD generates one or two new individuals, it decides whether to use mutation or recombination. This is done probabilistically via the command line parameters APPLYMUTATIONPROB: with probability APPLYMUTATIONPROB VFD uses mutation, with probability 1-APPLYMUTATIONPROB it uses recombination.

Checking for diversity is done in ISPOPULATIONDIVERSE(). It finds the error of the best tree (the first in the population) and the worst tree (the last in the population) at lines 26 and 27 respectively. Diversity is then calculated via "error of worst tree - error of best tree"/ "error of best tree" at line 28, which is then compared to DIVERSITY_THRESHOLD, another parameter of VFD. If diversity drops below this threshold, diversity is considered to be lost (line 29).

The next function is INITPOPULATION(), which periodically reinserts diversity into the population. The entire population is cleared (line 17) and reinitialized

Algorithm 6.2 VFD continued

```
 1: function MUTATE(parent)
 2:     z ← randint [0,numElements(parent)−1]
 3:     newSubtree ← generateRandomTree()
 4:     parent→setSubtree(z, newSubtree)
 5: end function
 6:
 7: function RECOMBINE(parent1, parent2)
 8:     z1 ← randint [0,numElements(parent1)−1]
 9:     z2 ← randint [0,numElements(parent2)−1]
10:     subTree1 ← parent1→getSubtree(z1)
11:     subTree2 ← parent2→getSubtree(z2)
12:     parent1→setSubtree(z1, subtree2)
13:     parent2→setSubtree(z2, subtree1)
14: end function
15:
16: function INITPOPULATION( )
17:     population ← List()
18:     for k ← 0,...,MU−1 do
19:         population[k] ← generateRandomTree()
20:     end for
21:     setError(population)
22:     sort(population)
23: end function
24:
25: function ISPOPULATIONDIVERSE(population)
26:     min ← population[0]→getError()
27:     max ← population[MU−1]→getError()
28:     div ← (max-min)/min
29:     return div > DIVERSITY_THRESHOLD
30: end function
31:
32: function ISCONVERGED( )
33:     return population[0]→getError() < MIN_ERROR
34: end function
```

with randomly generated trees (lines 18–20). The final steps at lines 21 and 22 calculate the error of each tree and sort the population (on error). Readers familiar with GP most likely notice that VFD's treatment of diversity differs from common practice in GP. We added a paragraph on the reasons for this difference in Section 6.7.

The final function in Algorithm 6.2 is the ISCONVERGED() function, which determines whether the current best individual is good enough to allow stopping of VFD. If its error is lower than the threshold value MIN_ERROR (specified by the user), VFD stops.

6.4.4 Bloat in GP

When recombination exchanges, for instance, the root of the first tree with a leaf of the second tree, the second tree can increase in depth and in number of elements. Over time, this typically leads to large and deep trees, with negative effects on both speed and memory usage. This problem is called *bloat* and must be dealt with by VFD. It does this by enforcing a maximum on the number of elements in the tree, as specified by the command line parameter MAXELEMENTSINTREE. This feature is not shown in the MUTATE() and RECOMBINE() functions in Algorithm 6.2 to keep the listing readable, but it is present in the implementation of VFD. Additionally, the SORT() function, which sorts a given set of trees by error in ascending order, has a built-in preference for trees with a small number of elements. Specifically, if two trees have equal error, the sort function puts the tree with the fewest elements in front. This gives VFD a slight inclination to discover short trees and prevent bloat.

6.4.5 Parent selection and survivor selection

We continue with the SELECTPARENT() function in Algorithm 6.3, which is used by the mutation and recombination operators to determine which parent(s) to act upon. Following convention in the GP community, VFD relies on a strategy called *over-selection* when selecting parents. In this strategy the population is split into two groups, one containing 'good' parents and the other with 'bad parents'. The two groups are separated by taking the sorted population and defining the first 'GOODPCT' percent individuals as good parents, and the remaining trees as bad parents. The parameter GOODPCT is automatically determined by VFD from the size of the population MU. For this, VFD again relies on GP-conventions and uses values ranging from $4-32\%$, as described in [44, Table 6.4]. Once the split point z_1 is known (line 2), a parent is selected from the good parents with probability SELECTFROMGOODPROB and from the bad parents otherwise. SELECTFROMGOODPROB is set to 0.8, again following conventions in the GP community. The selection is done at lines 4 and 6. Note that for recombination the SELECTPARENT() function is called twice.

Algorithm 6.3 VFD continued

```
 1: function selectParent( )
 2:     z1 ← floor(MU·GOODPCT)
 3:     if select from good then
 4:         z2 ← randint[0, z1 − 1]
 5:     else
 6:         z2 ← randint[z1, MU−1]
 7:     end if
 8:     return population[z2]
 9: end function
10:
11: function survivorSelection(population)
12:     remove population[MU:MU+LAMBDA−1]
13: end function
14:
15: function setError(trees)
16:     for tree in trees do
17:         maxError ← 0
18:         for q ← 0, . . . , Q − 1 do
19:             err ← calcError(samplePointSets[q], tree)
20:             maxError ← max (err, maxError)
21:         end for
22:         tree→setError(maxError)
23:     end for
24: end function
```

The survivorSelection() function is used by VFD in each generation after the LAMBDA children have been generated. Its purpose is to select MU survivors from among the MU+LAMBDA individuals currently in the population. VFD uses a greedy approach and simply removes the LAMBDA individuals with the worst error from the population (line 12).

6.4.6 Creating random trees

Mutation and initialization of the population use function generateRandomTree() for creating new trees. The type of operator in an internal node is determined randomly: it is a '+', '-', '*', or '/' with probability PROB_PLUS, PROB_MIN, PROB_MULTIPLY, and 1-PROB_PLUS-PROB_MIN-PROB_MULTIPLY, respectively. Similarly, leaf nodes are a variable, MDP pa-

rameter, or constant with probability PROB_VARIABLE, PROB_PARAMETER, and 1-PROB_VARIABLE-PROB_PARAMETER. We expect that the division operator '/' is needed less often than the others, and this is reflected in the values of VFD's parameters in the bottom part of Table 6.1.

6.4.7 Goodness of fit (error)

So far we have not yet discussed how the error of a tree is defined. This definition ties the GP approach of VFD to the MDP setting of finding a good relative value function. The error of a tree must be chosen in such a way that a low error corresponds to a good fit of the function described by the tree on the sample points obtained from the MDP. For VFD the error $\mathcal{E}_q$ on sample point set S_q is calculated via

$$\mathcal{E}_q = \max_{(s,V(s))\in\mathcal{S}_q} \frac{|\widetilde{V}(s) - V(s)|}{V(s)}. \tag{6.1}$$

Here, $\widetilde{V}(\cdot)$ is the function discovered by VFD and $V(\cdot)$ the optimal relative value function found by value iteration. The error $\mathcal{E}_q$ is calculated in the function CALCERROR() at line 19 in Algorithm 6.3. The error of a tree is then defined as

$$\mathcal{E} = \max_{q\in[0,Q-1]} \mathcal{E}_q, \tag{6.2}$$

i.e., the error of the tree is its worst error achieved on all the sample point sets. The error in Eq. (6.1) uses a relative measure of error by dividing by $V(s)$, contrary to, e.g., the mean squared error. This ensures that sample points that naturally have large values for $V(s)$ do not dominate the search process of VFD. Also, we use "$\max_{(s,V(s))\in\mathcal{S}_q}$" rather than "$\text{mean}_{(s,V(s))\in\mathcal{S}_q}$" (i.e., MAPE). With MAPE, a large relative error for a small sample point s can be mitigated by a small relative error of large sample points. In the context of MDPs, however, small states are usually visited more often, so we require a better fitting relative value function in such states. At larger states we want to allow larger errors. Therefore, using "$\max_{(s,V(s))\in\mathcal{S}_q}$" in VFD is preferable to MAPE.

Parameters Name	In example	Allowed values
Command line		
SEED	3,151,492	[0,MAXINT]
MU	1,000	[1,MAXINT]
LAMBDA	500	[1,MAXINT]
MAXELEMENTSINTREE	125	[1,MAXINT]
MIN_ERROR	0.2	[0,1]
APPLYMUTATIONPROB	0.2	[0,1]
DIVERSITY_THRESHOLD	0.01	[0,MAXDOUBLE]
Parent selection		
GOODPCT	0.32	[0,1]
SELECTFROMGOODPROB	0.8	[0,1]
Random tree creation		
PROB_PLUS	0.3	[0,1]
PROB_MINUS	0.3	[0,1]
PROB_MULTIPLY	0.3	[0,1]
PROB_PARAMETER	0.45	[0,1]
PROB_VARIABLE	0.45	[0,1]

TABLE 6.1: The parameters available to VFD (first column), the values assigned to them for the example MDP in Section 6.5 (second column), and the values allowed by VFD (third column).

6.5 Example MDP

In the following paragraphs we illustrate VFD on an example MDP. We describe how VFD is configured and, in doing so, we have to choose the command line parameters of VFD, listed in the top part of Table 6.1. It shows the values we are going to choose in this section, as well as the range of values that is allowed. For completeness, it also has the parameters that were discussed in Section 6.4 and the values assigned to them by VFD. Before starting, we emphasize that we make reasonable choices for VFD's parameters and the sample point sets, rather than seeking choices that lead to, e.g., fast run times. Our focus is on demonstrating that VFD can indeed be used to learn a relative value function that yields a near-optimal policy.

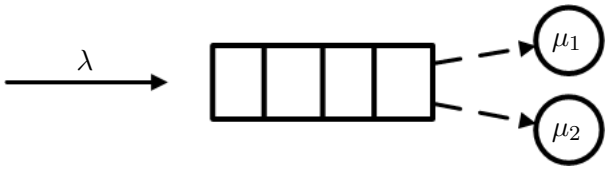


FIGURE 6.4: An M/M/2 system with control, where jobs (arriving with rate λ) from the queue have to be assigned to either a fast server S_1 (with service rate μ_1) or to a slow server S_2 (with service rate $\mu_2 < \mu_1$).

The MDP in this section is suitable for demonstrating VFD because:

- No known expression for the optimal policy or the relative value function exists, so we have no prior knowledge that VFD can capture the optimal relative value function.
- The system resembles a combination of an $M/M/1$ and $M/M/2$ system, which helps us when generating sample point sets and when choosing MAXELEMENTSINTREE.
- The system is relatively simple and easy to understand.
- The state space is small, which keeps run times of VFD short.

6.5.1 Model formulation

Figure 6.4 shows a queue with Poisson arrivals (rate λ) and two servers with exponential service rates μ_1 and μ_2 (without loss of generality we take the service rates such that $\mu_1 > \mu_2$). Arriving jobs are put into the queue and, when they reach the head of the queue, have to be assigned non-preemptively to either the fast server (S_1) or the slower server (S_2). This decision is taken after a job completion, as well as when a new job arrives at the queue. We model this scenario as an MDP, with states $(x, i) \in \mathcal{X} = \mathbb{N} \times \{0, 1\}$. Here, x denotes the number of jobs in the queue and at S_1, and i the number of jobs at S_2. Our aim is to minimize the average number of jobs in the system. From [81] we have the optimality equation

$$g + V(x,i) = x + i + \lambda W(x+1,i) + \mu_1 W((x-1)^+,i) + \mu_2 W(x,0) \qquad (6.3)$$

with

$$\begin{aligned} W(x,0) &= \min\{V(x,0); V(x-1,1)\} \quad \text{if } x > 0, \\ W(0,i) &= V(0,i), \\ W(x,1) &= V(x,1). \end{aligned} \qquad (6.4)$$

The function $W(x,i)$ reflects the decision to be taken after the occurrence of an event. In particular, if S_2 is empty the decision is between leaving the job in the queue ($V(x,0)$) or moving one job from the queue to S_2 ($V(x-1,1)$), as shown in Eq. (6.4). If the queue and S_1 are empty then moving a job is not possible and the state of the system does not change ($W(0,i) = V(0,i)$). Also, if the second server is busy the state does not change ($W(x,1) = V(x,1)$). In Eq. (6.3), $x+i$ reflects the number of jobs in the system, $W(x+1,i)$ the decision upon a job arrival, $W((x-1)^+,i)$ the decision when a job is completed at S_1, and $W(x,0)$ the decision when a job is completed at S_2. Finally, the constant g is the time-average cost of the system.

Note that this formulation allows preemptive behavior, since the expression $W(1,0) = \min\{V(1,0); V(0,1)\}$ can result in moving a job in service at S_1 to S_2. However, since $\mu_1 > \mu_2$ and rates are exponential, such a move would result in a longer expected service time for the job than when it is left at S_1. Hence, the optimal policy automatically enforces non-preemptive behavior. Finally, in Eq. (6.3) and (6.4) we assume that the parameters are normalized such that $\lambda + \mu_1 + \mu_2 = 1$.

6.5.2 Generating sample point sets

The first step to running VFD is preparing the sample point sets. The steps were described in Section 6.4.1 and we repeat them here for convenience. In these steps, we include the fact that $m = 3$ for the example MDP, and that we use value iteration for generating the sample points.

1. Fix values for each of the three parameters λ, μ_1, and μ_2.
2. Run value iteration for the MDP.
3. Select sample points that capture the shape of the relative value function.
4. Save these sample points into a file.
5. Repeat steps 1–4 for Q pairs of the three parameters.

First we decide upon the number of sample point sets Q (for step 5) that we will generate, and on the MDP parameter values used for each set (for step 1). We make our choice for a worst-case scenario where S_2 is never used (i.e., an $M/M/1$ system) and choose parameters for the sample point sets based on the load $\rho = \lambda/\mu_1 \in [0,1]$. Then, we generate parameters μ_1 and μ_2 uniformly from $[0,1]$, set $\lambda_1 = \rho\mu_1$, at the same time ensuring that $\mu_1 > \mu_2$ and that $\lambda + \mu_1 + \mu_2 = 1$.

Set	ρ	λ	μ_1	μ_2
0	0.100	0.0814	0.8135	0.1051
1	0.400	0.2688	0.6719	0.0594
2	0.525	0.3158	0.6015	0.0827
3	0.650	0.3701	0.5693	0.0606
4	0.775	0.4028	0.5198	0.0774
5	0.900	0.4662	0.5180	0.0159
6	0.950	0.4804	0.5057	0.0139

TABLE 6.2: Model parameters per sample point set.

In the region $0 \leq \rho \leq 0.4$ the load on the system is low, and possible wrong decisions in a policy have little impact. Hence, we expect that an accurate relative value function in that region is not required, and we cover it by just two sample point sets: one at $\rho = 0.1$ and another at $\rho = 0.4$. Following similar reasoning, we choose two sample point sets 'close together' at $\rho = 0.9$ and $\rho = 0.95$ to cover scenarios with a high load. The region $0.4 < \rho < 0.9$ is then covered by $Q - 4$ sample point sets distributed evenly over the interval. Short experiments suggest that $Q = 7$ is a reasonable choice. The resulting ρ-values are $\{0.1, 0.4, 0.525, 0.65, 0.775, 0.9, 0.95\}$, and the model parameters of each set are listed in Table 6.2.

Note that, generally speaking, using many sample point sets (i.e., a large Q) ensures that VFD discovers a well-fitting relative value function. On the other hand, the points in each sample point set are used many times to evaluate trees, contributing significantly to the computational complexity. Moreover, VFD has to discover a relative value function that closely fits each sample point set, so using many sets increases the time needed by VFD to discover such a function. Consequently, choosing Q is a trade-off between the goodness of fit of the discovered relative value function, and the run time of VFD.

Now that the number of sets is chosen, the sample points in each set can be selected. To run value iteration we must decide on a boundary for the first dimension of the state space $\mathcal{X} = \mathbb{N} \times \{0, 1\}$. We use a value L to limit the state space to $\hat{\mathcal{X}} = [0, L] \times \{0, 1\}$, where L is the smallest value such that $\mathbb{P}(x > L) < 0.001$ in the worst case $M/M/1$ scenario. For each sample point set we then take 2×10 points, with the ten x-values evenly distributed over $[0, 0.75 \cdot L]$ and i both 0 and 1 (recall that $(x, i) \in \hat{\mathcal{X}}$ is a point in the state space). These sample points capture the shape of the relative value function and avoid boundary effects of value iteration (by using $\lceil 0.75 \cdot L \rceil$ instead of L). If $0.75 \cdot L < 10$ then we take only $\lceil 0.75 \cdot L \rceil$ points instead of ten. Finally,

we stop value iteration once the span of two consecutive iterations is less than 10^{-6}.

With these sample points, the part of the state space outside $\hat{\mathcal{X}}$ is not covered by sample points. Most likely, VFD will not discover a relative value function that extrapolates well outside $\hat{\mathcal{X}}$. By choosing L such that $\mathbb{P}(x > L) < 0.001$, we ensure that it is unlikely that the system reaches states outside $\hat{\mathcal{X}}$, thus minimizing the effect of VFD's inability to extrapolate. In general, when applying VFD the user should keep in mind that it is good at interpolating between sample points, and not at extrapolating. Hence, the sample points should cover the area in the state space that the user is most interested in. A similar argument holds for the placement of the Q sample point sets in the parameter space.

6.5.3 Determining command line parameters

The next step is determining the command line parameters, as listed in the top part of Table 6.1. The first, SEED, can be set to any desired integer value, as it is only used to initialize the random number generator. For the population size MU and the number of children LAMBDA we follow current trends in GP and choose them such that LAMBDA$<$MU. In [44] populations with several thousands of individuals are suggested, but since our MDP has fairly low dimensionality we conservatively set MU$=$ 1,000 and LAMBDA$=$ 500.

For the parameter MAXELEMENTSINTREE we manually count the number of elements needed for the $M/M/1$ relative value function (13) and the $M/M/2$ relative value function ($\approx$ 90), based on the expressions in [28]. Then, we set MAXELEMENTSINTREE to a value somewhat higher than 90 (125), and ran some short experiments to see how large the resulting trees where. These experiments suggest that using 125 elements is sufficient. In general it is wise to set MAXELEMENTSINTREE to a slightly bigger value than expected, since that gives VFD some more freedom. Also, the SORT() function prefers smaller trees, so this tends to counteract a possibly too large value of MAXELEMENTSINTREE.

Next is MIN_ERROR, which influences the stopping criterion of VFD. Large values for MIN_ERROR let VFD stop quickly (but with a badly fitting tree), smaller values allow VFD to search longer (with a better fitting tree). Note that for the current MDP the performance of a discovered relative value function depends on the decision $\min\{\widetilde{V}(x,0); \widetilde{V}(x-1,1)\}$. Even if $\widetilde{V}(x,i)$ is not highly accurate, the decision can still be correct. Hence, we choose MIN_ERROR quite large and set MIN_ERROR$=$ 0.20.

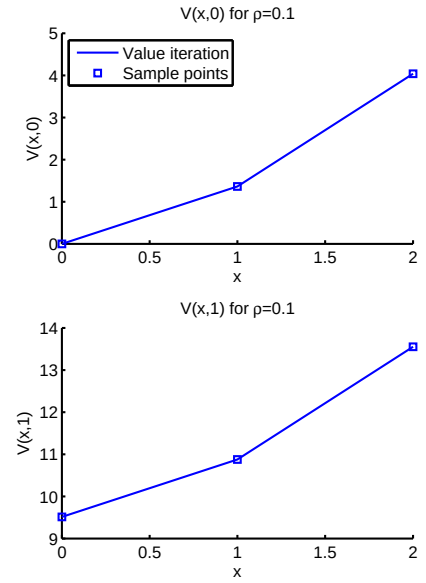
The value of DIVERSITY_THRESHOLD is determined by visually observing the progress made by VFD in terms of error in several short experimental runs. VFD should have sufficient time to discover good functions in between reinitialisations of the population, but should stop as soon as error stops decreasing significantly. This means that DIVERSITY_THRESHOLD should not be too high. After some experiments we set it to 0.01, i.e., diversity is lost when the worst tree differs by at most 1% from the best tree (in terms of error).

Parameter APPLYMUTATIONPROB is used by VFD to decide between using the mutation or recombination operators. The GP literature (see [44, Sec. 6.4] and the references therein) suggests using a mutation probability in the order of 0.05. However, experiments on the current MDP indicate that setting APPLYMUTATIONPROB to 0.2 yields better results.

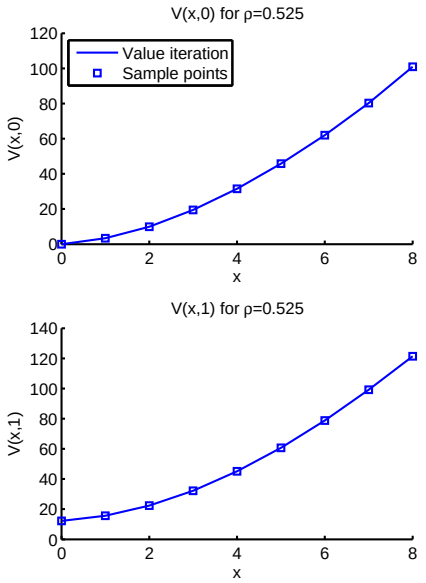
6.6 Numerical results

6.6.1 Sample points

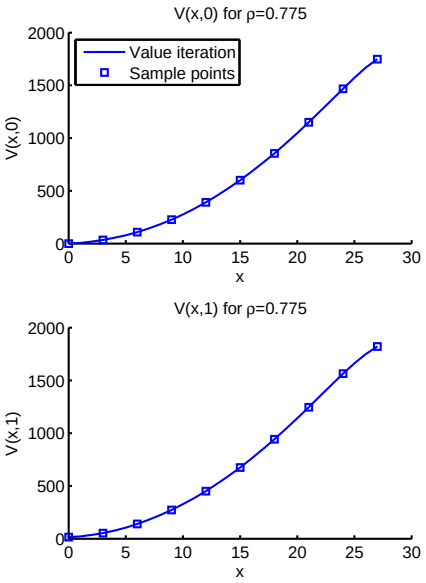
Section 6.5.2 describes how the sample points for our MDP example are generated. The values for the model parameters per sample point set are outlined in Table 6.2. For each of the model parameters in Table 6.2 we then run value iteration to find the sample points. Figure 6.5 shows the resulting sample points (marked by squares) for several of the sets. Note that the system with high load (Figure 6.5D) the optimal relative value function attains values in the order of 10^4, whereas for lower loads in Figures 6.5A and 6.5B these values are significantly smaller. Also, in Figure 6.5A the boundary $\lceil 0.75 \cdot L \rceil$ for value iteration is smaller than the number of desired sample points (ten), in which case only $\lceil 0.75 \cdot L \rceil$ sample points are retained. This results in three sample points for both $i = 0$ and $i = 1$, i.e., six sample points in total.



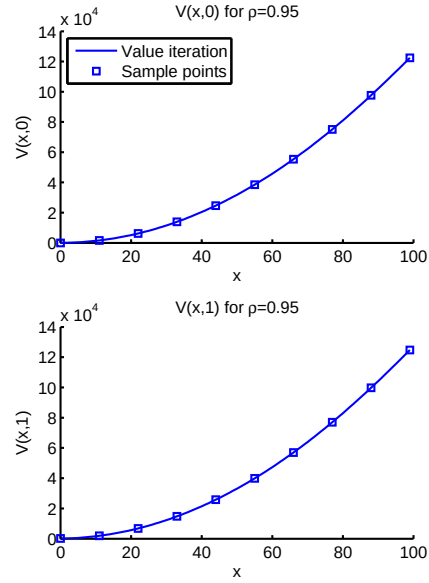
(A) Set 0.



(B) Set 2.

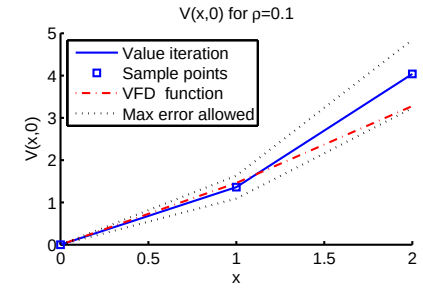


(C) Set 4.

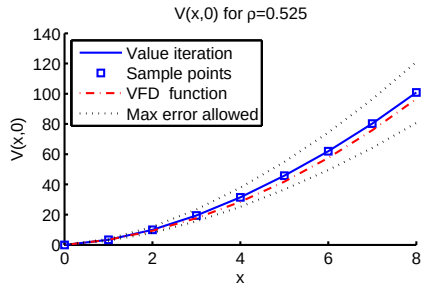


(D) Set 6.

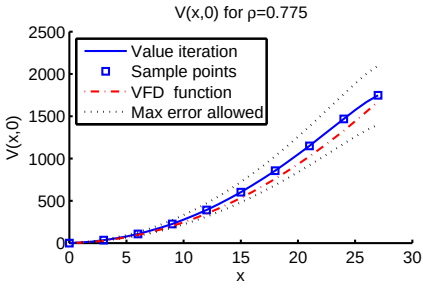
FIGURE 6.5: Sample point sets $0, 2, 4$, and 6.



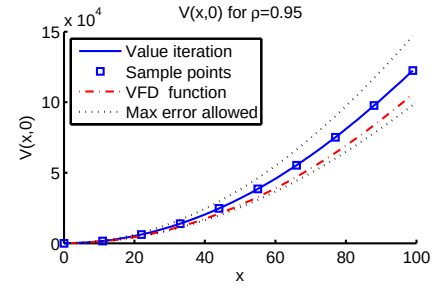
(A) Set 0.



(B) Set 2.



(C) Set 4.



(D) Set 6.

FIGURE 6.6: $\widetilde{V}(x,i)$ for sets $0, 2, 4$, and 6.

6.6.2 The discovered relative value function

Having specified all the input for VFD, it is ready to run. The relative value function $\widetilde{V}(x,i)$ discovered by VFD is

$$\begin{aligned}\widetilde{V}(x,i) = i/\Big[&0.28\mu_2(2\lambda\mu_2(i+\mu_1)(2\lambda+\mu_1)-i+\mu_2)\\ &\cdot\left((i+\lambda)\left(\frac{\lambda^2}{\mu_1}+\mu_2\right)+i-\mu_1\right)+\mu_2\Big]+x\\ &-\lambda\left(\lambda^2+1\right)x\Big[\lambda^2-\lambda\frac{\lambda^2\left(\frac{3.58i\lambda}{\mu_1}+3.58\lambda^2x+x\right)+\mu_2x}{\mu_2}\\ &-3.58(\lambda+\mu_1)-3.58\lambda x-\mu_1x-2\mu_2-x\Big].\end{aligned} \tag{6.5}$$

$\widetilde{V}(x,i)$ is plotted in Figure 6.6 (dash-dotted line) together with the sample points for the same sets as in Figure 6.5. Additionally, the figure contains two lines (dotted) above and below the sample points that indicate how much $\widetilde{V}(x,i)$ is allowed to differ from the sample points, as specified by the error criterion in Eq. (6.1) and by the parameter MIN_ERROR. When running, VFD continues looking for a relative value function until one is found that lies completely between this upper and lower bound. Figure 6.6 demonstrates that $\widetilde{V}(x,i)$ resembles $V(x,i)$ well, and that it indeed lies between the specified bounds. By modifying the parameter MIN_ERROR, the user of VFD can control the distance between the upper and lower bounds, and thus the accuracy of $\widetilde{V}(x,i)$. Also, observe that the distance between the upper and lower bound increases as x gets larger, as a consequence of our choice for a relative error criterion (as discussed in Section 6.4.7).

6.6.3 The policy derived from $\widetilde{V}(x,i)$

Next, we convert $\widetilde{V}(x,i)$ to a (algebraic) policy using a technique called *one-step policy improvement*, introduced in [118]. Observe that for states $(x,1)$ it is not possible to assign a job to server S_2, so the policy is trivial in these states. Therefore, we focus on states $(x,0)$. To obtain the policy, we take the term $\min\{V(x,0); V(x-1,1)\}$ in Eq. (6.4) and substitute $\widetilde{V}(x,i)$ for $V(x,i)$. Evaluating the minimum results in an action for each state $(x,0)$, i.e., server S_2 is used when $\widetilde{V}(x,0) > \widetilde{V}(x-1,1)$. Unfortunately, the resulting inequality

Set	ρ	g	$\tilde{g}$	Policy
0	0.100	0.1107	0.1107	Use S_2 if $x > 25.5719$
1	0.400	0.6643	0.6643	Use S_2 if $x > 10.2648$
2	0.525	1.0589	1.0665	Use S_2 if $x > 5.9715$
3	0.650	1.7107	1.7368	Use S_2 if $x > 6.4751$
4	0.775	2.4684	2.5085	Use S_2 if $x > 4.6315$
5	0.900	7.3973	7.7279	Use S_2 if $x > 15.5992$
6	0.950	12.8241	13.5369	Use S_2 if $x > 17.4802$

TABLE 6.3: time-average cost $\tilde{g}$ for the policy based on the relative value function in Eq. (6.5) discovered by VFD. These costs are compared to costs g of the optimal policy. The policy in the last column indicates for which states $(x, 0)$ a job should be assigned to server S_2.

is lengthy and challenging to interpret. Instead, we simplify the inequality for parameters λ, μ_1, μ_2 of the sample point sets in Table 6.2, and list the policies in the last columns of Table 6.3. The policies indicate for which states $(x, 0)$ the second server S_2 should be used. All policies are of threshold type, and the same structure holds for the optimal policy (see [81] for a proof). The time-average cost $\tilde{g}$ of the discovered are in Table 6.3, and demonstrate that the policy yields good results for the various model parameter values.

6.6.4 VFD and interpolation

The time-average cost in Table 6.3 are based on the model parameters in Table 6.2, which were given to VFD as input. As mentioned in Section 6.5.2, we expect that VFD is able to interpolate well in the range $[0.100, 0.950]$ for ρ. To investigate this, we fix new values for ρ within that range (the second column in Table 6.4) and generate new values for the model parameters λ, μ_1, and μ_2 (columns $3-5$). Then, we rerun value iteration to get the costs g of the optimal policy, and apply policy evaluation to find the costs $\tilde{g}$ of the policy based on $\widetilde{V}(x, i)$ from Eq. (6.5). The last two columns of Table 6.4 demonstrate that g and $\tilde{g}$ are consistently close and that VFD performs well on these new model parameters. We repeated this experiment several times for other values of the parameter SEED, and VFD continually yielded similar good results.

Set	ρ	λ	μ_1	μ_2	g	$\tilde{g}$
0	0.010	0.0088	0.8832	0.1080	0.0101	0.0101
1	0.200	0.1533	0.7663	0.0805	0.2496	0.2496
2	0.300	0.2094	0.6981	0.0924	0.4270	0.4270
3	0.450	0.2848	0.6329	0.0823	0.8067	0.8100
4	0.600	0.3686	0.6143	0.0171	1.4930	1.4930
5	0.700	0.3823	0.5462	0.0715	1.9669	2.0080
6	0.825	0.4443	0.5385	0.0172	4.3761	4.4744
7	0.875	0.4567	0.5219	0.0215	5.7497	5.9840
8	0.925	0.4571	0.4942	0.0487	5.8536	6.0514

TABLE 6.4: time-average cost $\tilde{g}$ for the policy based on the relative value function discovered by VFD, compared to costs g of the optimal policy. The model parameters (λ, μ_1, μ_2) and loads (ρ) are different from the ones VFD was given as input.

6.6.5 Computational complexity

With the model parameter values from Table 6.2 and the corresponding sample point sets VFD requires 2 minutes and 7 seconds to discover the $\widetilde{V}(x,i)$ from Eq. (6.5). Since VFD relies on several sources of randomness (controlled via command line parameter SEED), we inspect whether this run time is representative of VFD in general. To this end, we run VFD for 25 different values of SEED, record the run times, and compute the median of these run times. This results in a median run time of 2 minutes and 21 seconds, which corresponds well with the previously observed run time. For the MDP in this chapter the run time is quite short, which is mainly due to the small state space of the MDP in Eq. (6.3). On MDPs with larger state spaces the run time will be longer, but we feel that this is well worth the effort. Obtaining near-optimal policies for large MDPs via mathematical procedures is extremely challenging, time consuming, and does not always yield results. VFD, however, is easy to set up and run.

6.6.6 VFD applied to $M/M/1$

In Section 6.2 we claimed that for MDPs that allow for an explicit closed-form expression of the optimal relative value function, VFD can find this optimal relative value function. As an illustration, we let VFD discover the relative value function of an $M/M/1$ queue. To this end, we set $\mu_2 = 0$, regenerate the sample

point sets, and run VFD with parameter MIN_ERROR set to 0.0001 (slightly bigger than 0 to allow for small numerical inaccuracies in value iteration). VFD discovers the function

$$\widetilde{V}(x) = \frac{x(\lambda + \mu + x)}{-2\lambda + 2\mu},$$

which simplifies to

$$\widetilde{V}(x) = \frac{x(x+1)}{2(\mu - \lambda)}.$$

This is indeed the relative value function of an $M/M/1$ queue [28], and demonstrates that VFD is able to discover the closed-form expression that we expected.

6.7 Discussion

The results from the previous section demonstrate that VFD is able to discover relative value functions that closely resemble the optimal relative value function, and that the policy derived from a discovered relative value function perform wells. The good results in this chapter indicate that VFD is a promising technique and shows great potential. The research on VFD so far is, however, an initial step of exploring the idea of combining GP with MDPs. In particular, using VFD to gain valuable insights into the structure of an optimal relative value function is still unexplored. Another interesting application scenario of VFD is that of the control of an MDP with time-varying parameters. Having an algebraic policy prevents the need to make and analyze a time-dependent model. In the remainder of this section we list several potential directions for future research.

Shorter descriptions. In this chapter we showed the relative value function discovered by VFD in Eq. (6.5), but we did not analyze it further. It can, however, provide useful insights. For instance, $\widetilde{V}(x, i)$ in Eq. (6.5) contains the element λ/μ_1, the load of an $M/M/1$ system. It does, however, not contain $\frac{\lambda}{\mu_1+\mu_2}$, the load on an $M/M/2$ system. At the moment it is quite difficult to interpret the discovered relative value function, because the expression in Eq. (6.5) is somewhat long. We even expect that it is acceptable to sacrifice some accuracy in return for shorter trees. An inclusion of this feature in VFD might help in discovering relative value functions that are simpler and easier to interpret.

Include prior knowledge. VFD does not utilize any prior knowledge about the structure of the relative value function in the population. However, it might speed up the search process or result in better relative value functions if this knowledge is included. For the MDP in this chapter, we could for instance add several elements of the $M/M/1$ and $M/M/2$ relative value function to the population, such as λ/μ_1 , $\lambda/(\mu_1+\mu_2)$, and x^2 (both the $M/M/1$ and $M/M/2$ relative value functions are quadratic in x).

Different error criterion for large MDPs. Prior to running VFD, it has to be supplied with sample point sets. For the MDP in this chapter, each set contains several points $((x,i),V(x,i))$ that together capture the shape of the optimal relative value function. The $V(x,i)$ are obtained by value iteration, which is computationally feasible for the small example MDP. For larger MDPs, however, this might not be possible. Earlier, we already mentioned the use of TD-learning as an alternative to value iteration. Another potential solution is to determine the error of a tree using the Bellman error of the optimality equation of the MDP, instead of with Eq. (6.1). For the MDP in this chapter, the modified error of a tree would be

$$\mathcal{E}_q = \sum_{(x,i)\in\mathcal{S}_q} \beta^{x+i} \Big| -\widetilde{V}(x,i) + x + i + \lambda\widetilde{W}(x+1,i) + \mu_1\widetilde{W}((x-1)^+,i) + \mu_2\widetilde{W}(x,0)\Big|. \tag{6.6}$$

This expression is based on the optimality equation in Eq. (6.3) and Eq. (6.4), with $V(x,i)$ replaced by $\widetilde{V}(x,i)$. Note that $\mathcal{E}_q \geq g$, because that is the error reached by the optimal relative value function. Also, the sample point sets $\mathcal{S}_q$ no longer contain $V(x,i)$, only a number of points (x,i) in the state space. Finally, to avoid overfitting to sample points farther away in the state space, the sum is weighted by a factor β^{x+i}, with β a suitable constant (in Eq. (6.1) overfitting was avoided by dividing by $V(x,i)$, but in this modified error criterion $V(x,i)$ is no longer available). We expect that with this modification, VFD is better able to handle large MDPs.

Additional operators. The current version of VFD uses only operators $\{/,*,+,-\}$, but the representation of a function in GP is flexible enough to also allow for, e.g., exponents, square roots, logarithms, and rounding. Additionally, we could add other genetic operators besides mutation and recombination, such as deleting and inserting nodes.

Determining VFD's parameters. In Section 6.5 we determined values for the parameters of VFD. We wanted to set the parameters of VFD to values that

yield good policies. In particular, we were not looking for the best parameter settings. The current, basic, MDP does not require too much consideration for the VFD parameters, but for larger systems we expect the parameter values to be more important. A potential improvement is to use a parameter tuning tool such as Bonesa [143] to select good parameters, or to learn parameters on the go with, e.g., a co-evolutionary algorithm (see [47] for an example).

Improve diversity handling. The current setup of VFD reinitializes the entire population when diversity is lost, so it does not attempt to maintain diversity of a population. Upon loss of diversity the search is simply restarted elsewhere. With the basic MDP we used in this chapter, such a naive attitude towards diversity is sufficient to get a good relative value function quickly. However, for MDPs with larger state spaces, or MDPs that require a smaller error, this approach most likely does not yield a sufficiently good relative value function in a reasonable amount of time. Traditionally, GP algorithms employ a diversity maintenance scheme, e.g., a temporary increase of APPLYMUTATIONPROB upon loss of diversity. We expect that VFD will also need a diversity maintenance strategy, as we continue our experiments with VFD in the near future. For the current chapter we decided not to include such a scheme, because that would have resulted in even more parameters for VFD. This would have clouded our focus on discovery of relative value functions and the resulting policies in the context of MDPs.

Learning policies. With certain MDPs it is also possible to use VFD to learn policies directly. In [81] the author proves that the optimal policy for the MDP in this chapter is a switching curve, i.e., there is a threshold T such that only S_1 is used for $x \leq T$ and both S_1 and S_2 are used for $x > T$. We can thus apply VFD to sample points of this threshold T and learn an expression for T in terms of the model parameters. This experiment is the topic of the next chapter. Another example is the improved policy $\pi'(i, j, N)$ from Chapter 5, shown in Eq. (5.26). This policy includes a parameter $\hat{\alpha}$ that is determined with a numerical procedure. Instead of this procedure, VFD can be applied to sample points of $g'(\alpha)$ and thus help discover an expression for $g'(\alpha)$, which can then be minimized with respect to α. This yields an expression for the parameter $\hat{\alpha}$ in terms of the model parameters $\lambda_1, \lambda_2, \mu_1$, and μ_2. With this expression for $\hat{\alpha}$, the policy $\pi'(i, j, N)$ no longer needs a numerical procedure.

Consistency of discovered value functions. VFD is a stochastic process, and running VFD with different values for the parameter SEED should ideally result in the same discovered relative value function. This is particularly useful when analyzing the structure of the value function discovered by VFD in order to learn something about the true relative value function. We expect that such

consistency is difficult to achieve, because VFD imposes no restrictions on the description of a discovered value function, and VFD is satisfied with any tree that has a sufficiently low error. Analyzing the relative value function of an MDP is, however, still possible with VFD, by inspecting the discovered trees from several runs of VFD for common structures.

6.8 Conclusion

In this chapter we introduced VFD, a novel method for discovering algebraic descriptions of relative value functions of MDPs using a GP approach. We started with a description of GP, in particular of the representation used in GP, and of the mutation and recombination operators. Then we gave a detailed description of VFD, discussed an example MDP, and applied VFD to that MDP to discover a relative value function. To illustrate how a discovered relative value function can be used, we obtained a policy from it via one-step policy improvement. Numerical experiments demonstrated that this policy has near-optimal performance, both for model parameters that VFD was given a priori, and for new parameters. We identified several opportunities for future research, containing both improvements to VFD and alternative applications of the algorithm.

7

Discovery of Structured Optimal Policies in Markov Decision Processes

In this chapter we continue work on VFD, the novel method for discovery of relative value functions for Markov Decision Processes that we introduced in Chapter 6. VFD discovers algebraic descriptions of relative value functions using ideas from the Evolutionary Algorithm field and, in particular, these descriptions include the model parameters of the MDP. We extend that work and demonstrate how additional information about the structure of the MDP can be included in VFD. For this we use the same example MDP as in Chapter 6, and include prior knowledge that the optimal policy is of threshold type. We let VFD learn an expression for this threshold in terms of the model parameters, and numerically inspect its performance. We demonstrate that this alternative use of VFD also yields near-optimal policies, illustrating that VFD is not restricted to learning relative value functions and can be applied more generally.

This chapter is based on the results presented in [7] and [8].

7.1 Introduction

We use the example MDP from Section 6.5 to illustrate this alternative application of VFD. For convenience we repeat the description of the MDP here. Figure 7.1 shows a queue with Poisson arrivals (rate λ) and two servers with exponential service rates μ_1 and μ_2 ($\mu_1 > \mu_2$). Arriving jobs are put into the queue and, when they reach the head of the queue, they have to be assigned non-preemptively to either the fast server (S_1) or the slower server (S_2). This decision is taken after a job completion, as well as after a job arrival. The state space of the resulting MDP is $\mathcal{X} = \mathbb{N} \times \{0, 1\}$, where a state $(x, i) \in \mathcal{X}$ reflects that there are x jobs in the queue and at S_1, and i jobs at S_2. From [81] we have the optimality equation

$$g + V(x,i) = x + i + \lambda W(x+1,i) + \mu_1 W((x-1)^+,i) + \mu_2 W(x,0) \qquad (7.1)$$

with

$$\begin{aligned} W(x,0) &= \min\{V(x,0); V(x-1,1)\} \quad \text{if } x > 0, \\ W(0,i) &= V(0,i), \\ W(x,1) &= V(x,1). \end{aligned} \qquad (7.2)$$

The function $W(x,i)$ reflects the decision to be taken after the occurrence of an event. In particular, if S_2 is empty the decision is between leaving the job in the queue ($V(x,0)$) or moving one job from the queue to S_2 ($V(x-1,1)$), as shown in Eq. (7.2). If the queue and S_1 are empty then moving a job is not possible and the state of the system does not change ($W(0,i) = V(0,i)$). Also, if the second server is busy the state does not change ($W(x,1) = V(x,1)$). In Eq. (7.1), the second, third, and fourth term on the right-hand side correspond to the decision upon a job arrival, a job completion at S_1, and a job completion at S_2, respectively. Finally, the constant g is the time-average cost.

For the MDP in Eq. (7.1), we know that the optimal policy of Eq. (7.1) is of threshold type (see [81] for a proof). To be precise, for given model parameters λ, μ_1, μ_2 the optimal policy states that server S_2 should be used whenever $x > T$, i.e., when x exceeds a certain threshold T. By viewing this threshold as a function of the model parameters, we can apply VFD and discover an algebraic expression for T in terms of the model parameters (λ, μ_1, μ_2). Compared to using VFD for discovering relative value functions, this alternative application has the advantage that it is much faster, since the threshold T depends only on model parameters (λ, μ_1, μ_2) and not on state (x, i). Most importantly though, it demonstrates that VFD is flexible and not restricted to learning relative value functions.

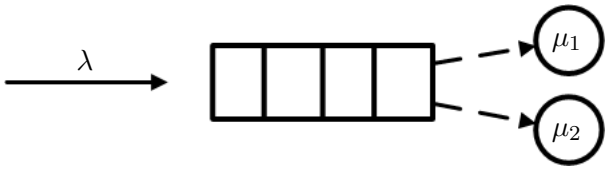


FIGURE 7.1: A queueing system with jobs arriving at rate λ, and with two servers S_1 (at the top) and S_2 (at the bottom). Both servers handle jobs from the queue, but server S_1 is faster than S_2 (reflected by service rates μ_1 and μ_2 such that $\mu_1 > \mu_2$). Upon completion of a job, and upon arrival of a new job, a controller decides whether to assign a job from the queue to either S_1 or S_2 (dashed line). See also Figure 6.4 and the description in Section 6.5.

The remainder of this chapter is structured as follows. First, in Section 7.2.1 we describe the setup of the experiments in this chapter. This includes the choice of sample points, and of the value of VFD's parameters. Section 7.3 demonstrates the results of applying VFD to discovery of thresholds, and Section 7.4 has concluding remarks.

7.2 Setup of experiments

In this section we describe the experiment where VFD learns an expression for the threshold T. Prior to running VFD, we make a minor change to VFD to facilitate the fact that a threshold has only integer values. Also, we determine the contents and location of the sample point sets, and VFD's parameters. As in Chapter 6, we make reasonable choices for the sample point sets and VFD's parameters, rather than seeking choices that lead to, e.g., the fastest run times. Our focus is on demonstrating that VFD can indeed be used to learn an algebraic description of a threshold policy that has near-optimal performance.

7.2.1 Learning thresholds with VFD

We denote the threshold discovered by VFD as $\widetilde{T}$, following the notation of Chapter 6. Before running experiments on VFD, we make one small adjustment to the error criterion used by VFD in Eq. (6.1). The threshold T is an integer number, whereas running VFD unchanged yields a $\widetilde{T}$ that potentially returns non-integer values. Additionally, the threshold is independent of the state (x, i), so we do not need to take the maximum over sample points anymore.

To this end, we modify the error on a sample point set $\mathcal{E}_q$ from Eq. (6.1) to

$$\mathcal{E}_q = \frac{\left|\lfloor \widetilde{T} \rfloor - T\right|}{T}, \tag{7.3}$$

where $\lfloor \widetilde{T} \rfloor$ denotes the largest integer smaller than $\widetilde{T}$. Also, we included notation T and $\widetilde{T}$ to emphasize that VFD discovers thresholds in this chapter. The expression for the total error E remains the same as in Eq. (6.2). This change to VFD is purely for convenience to avoid scenarios where VFD spends time improving a threshold $\widetilde{T} = 5.3$ to $\widetilde{T} = 5$, even though the improvement has no effect on the resulting policy. VFD also works well when the threshold is considered as a decimal number, as we demonstrated in [8].

7.2.2 Sample point sets

Recall that, when learning relative value functions with VFD, each sample point set contains several sample points (x, i) on the relative value function $V(x, i)$ for fixed parameters λ, μ_1, μ_2. For the current problem we can suffice with a single point in each sample point set, namely the threshold value T found by value iteration. Note that by making this change we only affect the input to VFD, and not the algorithm itself.

As in Chapter 6 we base the choice for the parameters λ, μ_1, μ_2 corresponding to each sample point set on the load $\rho = \lambda/\mu_1 \in$ of a $M/M/1$ system. In the region $0 \leq \rho < 0.7$ the load on the system is low, and possible wrong decisions in a policy have little impact. Therefore, we focus the experiment in this chapter on the region $0.7 \leq \rho \leq 1$. We want to let VFD discover a function that is close to optimality, so we place the sample point sets close together at intervals of 0.025. This results in $Q = 11$ sets at $\rho = 0.700, 0.725, \ldots, 0.925, 0.950$. Then we generate parameters μ_1 and μ_2 uniformly from $[0, 1]$, and set $\lambda_1 = \rho\mu_1$. In generating these values we also ensure that $\mu_1 > \mu_2$ and that $\lambda + \mu_1 + \mu_2 = 1$. The resulting parameters are in Table 7.1. Recall from Chapter 6 that when applying VFD to MDPs with a large dimensionality, having many sample point sets is typically not computationally feasible. Each (large) sample point set is used with each tree in the population to determine how well that tree fits, so having many sample point sets increases VFD's run time severely. In this chapter, however, each sample point set contains just one point, so we can use many of them without affecting run time too much.

Set	ρ	λ	μ_1	μ_2
0	0.700	0.369	0.527	0.104
1	0.725	0.373	0.515	0.112
2	0.750	0.379	0.506	0.115
3	0.775	0.396	0.512	0.092
4	0.800	0.409	0.511	0.080
5	0.825	0.450	0.546	0.003
6	0.850	0.447	0.525	0.028
7	0.875	0.424	0.484	0.092
8	0.900	0.466	0.518	0.015
9	0.925	0.445	0.481	0.074
10	0.950	0.485	0.510	0.005

Table 7.1: Model parameters per sample point set.

Name	In example	Allowed values
SEED	3,151,492	[0,MAXINT]
MU	1,000	[1,MAXINT]
LAMBDA	500	[1,MAXINT]
MIN_ERROR	0	[0,1]
MAXELEMENTSINTREE	100	[1,MAXINT]
APPLYMUTATIONPROB	0.05	[0,1]
DIVERSITY_THRESHOLD	0.01	[0,MAXDOUBLE]

Table 7.2: The parameters available to VFD, the values assigned to them for the example MDP in Section 7.1, and the values allowed by VFD.

7.2.3 Setting parameters of VFD

Running VFD requires the specification of several parameters, listed in Table 7.2. Following the reasoning in Section 6.5.3 we keep parameters SEED, MU, LAMBDA, and DIVERSITY_THRESHOLD at the same value as in Table 6.1. Parameter MIN_ERROR we set to 0 because we want VFD to discover a perfectly fitting function on the sample point sets. For parameter APPLYMUTATIONPROB we rely on experience from the GP community and set it to 0.05. The remaining parameter, MAXELEMENTSINTREE, is determined with short experimental runs, suggesting that VFD is capable of finding well-fitting relative value functions with MAXELEMENTSINTREE= 100. The resulting values for the VFD parameters are shown in the second column of Table 7.2.

Set	ρ	T	$\widetilde{T}$
0	0.700	2	2
1	0.725	2	2
2	0.750	2	2
3	0.775	3	3
4	0.800	3	3
5	0.825	32	32
6	0.850	6	6
7	0.875	2	2
8	0.900	8	8
9	0.925	3	3
10	0.950	13	13

TABLE 7.3: Threshold $\widetilde{T}$ of the policy discovered by VFD, compared to threshold T of the optimal policy.

7.3 Numerical results

We run VFD with the sample point sets from Section 7.2.2 and the parameters from Section 7.2.3. The threshold $\widetilde{T}$ discovered by VFD is

$$\begin{aligned}\widetilde{T} = {} & 3\lambda - 2\mu_1 - 0.47 - \frac{\lambda - 0.12}{\lambda + \mu_1 - \lambda\mu_1 - 0.82} \\ & + \frac{\lambda}{\mu_2}(\mu_1 - 0.40)(\lambda - \mu_1\mu_2) \\ & - 0.47(\mu_1 - \lambda)\Big[\lambda + \mu_1 - \lambda(2\mu_1 - \mu_2) + 1 - \frac{\lambda + 2\mu_1}{\mu_2}\Big].\end{aligned} \tag{7.4}$$

In Table 7.3 we list the threshold $\widetilde{T}$ from VFD, and the threshold T of the optimal policy. The table demonstrates that the discovered expression for the threshold does indeed perfectly match the optimal values.

Next, we inspect the performance of the expression in Eq. (7.4) on model parameters that it was not trained on. To this end, we select 10 values for ρ, different from those in Table 7.3, and generate new parameters λ, μ_1, μ_2 as before. The new model parameters are shown in Table 7.4 in the first four columns. Then we calculate the threshold $\widetilde{T}$ (sixth column), and the optimal threshold T (fifth column). Table 7.4 demonstrates that the expression discovered by VFD yields thresholds that closely resemble the optimal thresholds,

ρ	λ	μ_1	μ_2	T	$\widetilde{T}$
0.710	0.411	0.578	0.011	17	17
0.735	0.421	0.573	0.007	25	25
0.760	0.422	0.555	0.023	8	8
0.785	0.429	0.546	0.025	7	7
0.810	0.443	0.548	0.009	15	15
0.835	0.416	0.498	0.086	2	3
0.860	0.429	0.499	0.072	3	3
0.885	0.458	0.517	0.025	6	6
0.910	0.474	0.521	0.004	18	19
0.935	0.465	0.497	0.038	4	4

TABLE 7.4: Threshold $\widetilde{T}$ of the policy discovered by VFD, compared to threshold T of the optimal policy. The model parameters are different from the ones VFD was given as input.

even for the new model parameters. We repeated this experiment several times, and the function discovered by VFD consistently returned thresholds close to optimal.

In order to investigate the run time of the alternative use of VFD described in this chapter, we repeat the process from Section 6.6.5. There, we recorded a median run time of 2 minutes and 21 seconds over 25 runs. In the current setup, running VFD 25 times gives a median run time of 47 seconds. This is faster than before, even with a low value for MIN_ERROR and many sample point sets ($Q = 11$) to cover the parameter space.

7.4 Conclusion

In this chapter we continued work on VFD, a novel algorithm for discovering relative value functions for Markov Decision Processes. We applied VFD to an example MDP, for which we know that the optimal policy is of threshold type. Instead of discovering the relative value function with VFD, we learned an algebraic expression for the threshold in terms of the model parameters. We numerically inspected the resulting threshold and compared the corresponding policy to the optimal policy. The results demonstrate that his alternative use of VFD also yields near-optimal policies and thresholds that closely resemble the optimal value.

Bibliography

Publications by the author

[1] D. Ma, M. Onderwater, F. Wetzels, G. J. Hoekstra, R. D. van der Mei, S. Bhulai, and L. Zhuang. Cost-efficient allocation of additional resources for the service placement problem in next-generation Internet. *Mathematical Problems in Engineering*, 2015:1–15, 2015.

[2] M. Mitici, M. Onderwater, M. de Graaf, J. van Ommeren, N. van Dijk, J. Goseling, and R. J. Boucherie. Optimal query assignment for wireless sensor networks. *International Journal of Electronics and Communications*, 69(8):1102 – 1112, 2015.

[3] M. Onderwater. An overview of centralised middleware components for sensor networks. To appear in *International Journal of Ad Hoc and Ubiquitous Computing*, 2015.

[4] M. Onderwater. Outlier preservation by dimensionality reduction techniques. *International Journal of Data Analysis Techniques and Strategies*, 7(3):231–252, 2015.

[5] M. Onderwater, S. Bhulai, and R. D. van der Mei. Value Function Discovery in Markov Decision Processes with Evolutionary Algorithms. To appear in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2015.

[6] M. Onderwater, S. Bhulai, and R. D. van der Mei. On the control of a queueing system with aging state information. *Stochastic Models*, 31(4): 588–617, 2015.

[7] M. Onderwater, S. Bhulai, and R. D. van der Mei. Discovery of structured optimal policies in Markov Decision Processes. *Under review*, 2015.

[8] M. Onderwater, S. Bhulai, and R. D. van der Mei. Learning optimal policies in Markov Decision Processes with Value Function Discovery. *Performance Evaluation Review*, 43(2):7–9, 2015.

[9] M. Onderwater, G. J. Hoekstra, and R. D. van der Mei. Throughput modeling of the IEEE MAC for sensor networks. *Under review*, 2015.

References

[10] Android sensor overview. `http://tinyurl.com/bl2qunc`.

[11] Indoor climate regulation for Dutch schools and kindergartens [in dutch]. `http://tinyurl.com/pt7hf7y`.

[12] Gartner's 2015 hype cycle for emerging technologies. `http://www.gartner.com/newsroom/id/3114217`.

[13] I. Adan and J. Resing. Queueing theory. `http://www.win.tue.nl/ iadan/queueing.pdf`, 2001.

[14] C. C. Aggarwal. *Data Streams: Models and Algorithms (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[15] Agilla. `http://mobilab.cse.wustl.edu/projects/agilla`, 2009.

[16] M. Agyemang, K. Barker, and R. Alhajj. A comprehensive survey of numeric and symbolic outlier mining techniques. *Intelligent Data Analysis*, 10(6):521–538, 2006.

[17] F. Aiello. MAPS: a mobile agent platform for Java Sun SPOTs. In *Proceedings of the 3rd International Workshop on Agent Technology for Sensor Networks*, pages 41–48, 2009.

[18] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[19] H. Alex, M. Kumar, and B. Shirazi. MidFusion: an adaptive middleware for information fusion in sensor network applications. *Information Fusion*, 9(3):332–343, 2008.

[20] P. Andreou, D. Zeinalipour-Yazti, M. Vassiliadou, P. K. Chrysanthis, and G. Samaras. KSpot: effectively monitoring the k most important events in a wireless sensor network. In *Proceedings of the 25th IEEE*

International Conference on Data Engineering, pages 1503–1506. IEEE, 2009.

[21] P. G. Andreou, D. Zeinalipour-Yazti, G. Samaras, and P. K. Chrysanthis. Towards a network-aware middleware for wireless sensor networks. In *Proceedings of the 8th International Workshop on Data Management for Sensor Networks*, 2011.

[22] AWARE. `http://grvc.us.es/aware/`, 2001.

[23] Q. Bai, S. Guru, D. Smith, Q. Liu, and A. Terhorst. A multi-agent view of the sensor web. *Advances in Practical Multi-Agent Systems*, pages 435–444, 2011.

[24] D. Barash. A genetic search in policy space for solving Markov decision processes. In *Proceedings of the AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, 1999.

[25] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4): 469–483, 1996.

[26] R. A. Becker, C. Volinsky, and A. R. Wilks. Fraud detection in telecommunications: History and lessons learned. *Technometrics*, 52(1):20–33, 2010.

[27] S. Bhulai. *Markov decision processes: the control of high-dimensional systems*. PhD thesis, VU University Amsterdam, Amsterdam, 2002.

[28] S. Bhulai and G. Koole. On the structure of value functions for threshold policies in queueing models. *Journal of Applied Probability*, pages 613–622, 2003.

[29] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on Selected Areas in Communications*, 18(3):535–547, 2000.

[30] T. Bleier et al. *SANY - an open service architecture for sensor networks*. The SANY Consortium, 2010.

[31] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[32] H. Cao, D. H. Hu, D. Shen, D. Jiang, J.-T. Sun, E. Chen, and Q. Yang. Context-aware query classification. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 3–10, 2009.

[33] M. A. Carreira-Perpinán. A review of dimension reduction techniques. *Technical Report CS-96-09 of the Department of Computer Science, University of Sheffield*, pages 1–69, 1997.

[34] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proceedings of the 26th VLDB Conference*, pages 89–100, 2000.

[35] H. S. Chang, H. Lee, M. C. Fu, and S. I. Marcus. Evolutionary policy iteration for solving Markov Decision Processes. *IEEE Transactions on Automatic Control*, 50(11):1804–1808, 2005.

[36] D. Chen and P. K. Varshney. QoS support in wireless sensor networks: A survey. In *Proceedings of the 2004 International Conference on Wireless Networks*, 2004.

[37] M. Compton, C. Henson, H. Neuhaus, L. Lefort, and A. Sheth. A survey of the semantic specification of sensors. In *Proceedings of the 2nd International Workshop on Semantic Sensor Networks, at the 8th International Semantic Web Conference*, volume 522, pages 17–32, 2009.

[38] M. Compton et al. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17(0):25–32, 2012.

[39] J. Cook. Why big data matters to Boeing, and what it means for your next flight. `http://tinyurl.com/nnzgh4h`, March 2015.

[40] Cougar. `http://www.cs.cornell.edu/bigreddata/cougar`, 2002.

[41] M. Dahlin. Interpreting stale load information. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1033–1047, 2000.

[42] J. J. Dai, L. Lieu, and D. Rocke. Dimension reduction for classification with gene expression microarray data. *Statistical Applications in Genetics and Molecular Biology*, 5(1):1–21, 2006.

[43] F. Duijm, A. Boerstra, C. W. J. Cox, W. van Doorn, T. Habets, and R. van Strien. Toetswaarden voor ventilatie in scholen en kindercentra [in Dutch]. *GGD Nederland, werkgroep binnenmilieu*, 2006.

[44] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin Heidelberg New York, 2003.

[45] H. J. Escalante. A comparison of outlier detection algorithms for machine learning. In *Proceedings of the Congreso Internacional en Computacion*, 2005.

[46] S. M. Fairgrieve, J. A. Makuch, and S. R. Falke. PULSENet^{TM}: an implementation of sensor web standards. In *Proceedings of the International Symposium on Collaborative Technologies and Systems*, pages 64–75. Northrop Grumman, CO, 2009.

[47] C. M. Fernandes, J. J. Merelo, and A. C. Rosa. Controlling the parameters of the particle swarm optimization with a self-organized criticality model. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature*, pages 153–163. Springer, 2012.

[48] I. Fodor. A survey of dimension reduction techniques. Technical Report UCRL-ID-148494, Lawrence Livermore National Laboratory, 2002.

[49] C. L. Fok, G. C. Roman, and C. Lu. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems*, 4(3):16:1–16:26, 2009.

[50] I. Galpin, C. Brenninkmeijer, F. Jabeen, A. Fernandes, and N. W. Paton. An architecture for query optimization in sensor networks. In *Proceedings of the 24th IEEE International Conference on Data Engineering*, pages 1439–1441. IEEE, 2008.

[51] J. Gama and M. M. Gaber. *Learning from Data Streams: Processing Techniques in Sensor Networks*. Springer Berlin Heidelberg New York, 2007.

[52] V. K. Garg and M. N. Murty. Feature subspace SVMs (FS-SVMs) for high dimensional handwritten digit recognition. *International Journal of Data Mining, Modelling and Management*, 1(4):411, 2009.

[53] C. Gearhart. Genetic programming as policy search in Markov Decision Processes. *Genetic Algorithms and Genetic Programming at Stanford*, pages 61–67, 2003.

[54] A. Ghodsi. Lecture notes of lecture 9 of the course "Data visualization" (STAT 442). Technical report, University of Waterloo, 2006.

[55] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: an architecture for a worldwide sensor web. *IEEE Pervasive Computing*, 2 (4):22–33, 2003.

[56] P. Gogoi, D. K. Bhattacharyya, B. Borah, and J. K. Kalita. A survey of outlier detection methods in network anomaly identification. *Computer Journal*, 54(4):570–588, 2011.

[57] M. Gorlatova, A. Wallwater, and G. Zussman. Networking low-power energy harvesting devices: Measurements and algorithms. In *Proceedings of the 30th IEEE International Conference on Computer Communications*, pages 1602 –1610, 2011.

[58] M. Gorlatova, M. Zapas, E. Xu, M. Bahlke, I. Kymissis, and G. Zussman. *CRAWDAD data set columbia/enhants (v. 2011-04-07)*. 2011. Published online at `http://crawdad.cs.dartmouth.edu/columbia/enhants`.

[59] R. Gupta and R. Kapoor. Comparison of graph–based methods for non–linear dimensionality reduction. *International Journal of Signal and Imaging Systems Engineering*, 5(2):101–109, 2012.

[60] S. Hadim and N. Mohamed. Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, 7(3):1–23, 2006.

[61] S. Hadim and N. Mohamed. Middleware for wireless sensor networks: A survey. In *Proceedings of the First International Conference on Communication System Software and Middleware*, pages 1–7. IEEE, 2006.

[62] L. W. Hahn, M. D. Ritchie, and J. H. Moore. Multifactor dimensionality reduction software for detecting gene–gene and gene–environment interactions. *Bioinformatics*, 19(3):376–382, 2003.

[63] W. K. Härdle and L. Simar. *Applied Multivariate Statistical Analysis*. Springer, Berlin, 2012.

[64] S. Harmeling, G. Dornhege, D. Tax, F. Meinecke, and K. Müller. From outliers to prototypes: Ordering data. *Neurocomputing*, 69(13–15):1608–1618, 2006.

[65] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18 (1):6–14, 2004.

[66] K. Henricksen and R. Robinson. A survey of middleware for sensor networks: state-of-the-art and future directions. In *Proceedings of the international workshop on Middleware for sensor networks*, MidSens '06, pages 60–65, New York, NY, USA, 2006.

[67] O. Hernández-Lerma and J. B. Lasserre. *Discrete-Time Markov Control Processes: Basic Optimality Criteria*. Springer, New York, 1996.

[68] G. Hinton and S. Roweis. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, volume 15, pages 857–864. MIT Press, 2002.

[69] Y. Ho, Q. Zhao, and Q. Jia. *Ordinal Optimization: Soft Optimization for Hard Problems*. Springer, Boston, MA, September 2007.

[70] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[71] J. Hu, M. C. Fu, V. R. Ramezani, and S. I. Marcus. An evolutionary random policy search algorithm for solving Markov Decision Processes. *INFORMS Journal on Computing*, 19(2):161–174, 2007.

[72] M. C. Huebscher and J. A. McCann. Adaptive middleware for context-aware applications in smart-homes. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, MPAC '04, pages 111–116, New York, NY, USA, 2004.

[73] IRISNet. `http://www.intel-iris.net`, 2003.

[74] R. Johnson and D. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, Upper Saddle River, New Jersey, 2002.

[75] K. K. Kandaswamy, G. Pugalenthi, K. Kalies, E. Hartmann, and T. Martinetz. EcmPred: prediction of extracellular matrix proteins based on random forest with maximum relevance minimum redundancy feature selection. *Journal of Theoretical Biology*, 317:377–383, 2013.

[76] A. Kansal, S. Nath, J. Liu, and F. Zhao. SenseWeb: an infrastructure for shared sensing. *IEEE MultiMedia*, 14:8–13, 2007.

[77] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi. Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1): 28–44, 2013.

[78] D. M. Kline and C. S. Galbraith. Performance analysis of the bayesian data reduction algorithm. *International Journal of Data Mining, Modelling and Management*, 1(3):223, 2009.

[79] T. Kohonen. *Self-Organizing Maps*. Springer Series in Information Sciences. Springer-Verlag Berlin Heidelberg, New York, Secaucus, NJ, USA, 3rd edition, 2001.

[80] I. Kononenko and I. Bratko. Information-based evaluation criterion for classifier's performance. *Machine Learning*, 6(1):67–80, 1991.

[81] G. Koole. A simple proof of the optimality of a threshold policy in a two-server queueing system. *Systems & Control Letters*, 26(5):301–303, 1995.

[82] KSPOT. `http://www.cs.ucy.ac.cy/ panic/kspot/`, 2009.

[83] B. Latré, P. Mil, I. Moerman, B. Dhoedt, P. Demeester, and N. van Dierdonck. Throughput and delay analysis of unslotted IEEE 802.15. 4. *Journal of Networks*, 1(1):20–28, 2006.

[84] J. Lattin, D. Carroll, and P. Green. *Analyzing Multivariate Data*. Thomson Brooks/Cole, Pacific Grove, CA, 2003.

[85] B. Lauwens, B. Scheers, and A. van de Capelle. Performance analysis of unslotted CSMA/CA in wireless networks. *Telecommunication Systems*, 44(1-2):109–123, 2010.

[86] D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *The Semantic Web–ISWC 2011*, pages 370–388. Springer, 2011.

[87] D. Le-Phuoc, H. N. Quoc, J. X. Parreira, and M. Hauswirth. The linked sensor middleware–connecting the real world and the semantic web. In *Proceedings of the 10th International Semantic Web Conference*, 2011.

[88] D. Le-Phuoc, H. Q. Nguyen-Mau, J. X. Parreira, and M. Hauswirth. A middleware framework for scalable management of linked streams. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16: 42–51, 2012.

[89] C. Y. Lee, H. I. Cho, G. U. Hwang, Y. Doh, and N. Park. Performance modeling and analysis of IEEE 802.15. 4 slotted CSMA/CA protocol with ACK mode. *International Journal of Electronics and Communications*, 65(2):123–131, 2011.

[90] P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, 37:85–95, 2002.

[91] K. Li. On principal Hessian directions for data visualization and dimension reduction: Another application of Stein's lemma. *Journal of the American Statistical Association*, 87(420):1025–1039, 1992.

[92] S. Li, Y. Lin, S. H. Son, J. A. Stankovic, and Y. Wei. Event detection services using data service middleware in distributed sensor networks. *Telecommunication Systems*, 26(2):351–368, 2004.

[93] Z. Lin, J. C. Bean, and C. C. White. A hybrid genetic/optimization algorithm for finite-horizon, partially observed Markov Decision Processes. *INFORMS Journal on Computing*, 16(1):27–38, 2004.

[94] H. Liu, T. Roeder, K. Walsh, R. Barr, and E. G. Sirer. Design and implementation of a single system image operating system for ad hoc networks. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, MobiSys '05, pages 149–162, New York, NY, USA, 2005. ACM.

[95] T. Liu and M. Martonosi. Impala: a middleware system for managing autonomic, parallel sensor systems. *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 38:107–118, 2003.

[96] LSM. `http://lsm.deri.ie/`, 2011.

[97] M. Maalouf and T. B. Trafalis. Rare events and imbalanced datasets: an overview. *International Journal of Data Mining, Modelling and Management*, 3(4):375, 2011.

[98] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[99] Magnet. `http://www.cs.cornell.edu/people/egs/magnetos/`, 2005.

[100] P. C. Mahalanobis. On the generalized distance in statistics. In *Proceedings of the national institute of sciences of India*, volume 2, pages 49–55, 1936.

[101] MAPS. `http://maps.deis.unical.it`, 2009.

[102] P. D. Marco, P. Park, C. Fischione, and K. H. Johansson. Analytical modeling of multi-hop IEEE 802.15.4 networks. *IEEE Transactions on Vehicular Technology*, 61(7):3191–3208, 2012.

[103] B. Martins, I. Anastácio, and P. Calado. A machine learning approach for resolving place references in text. In *Geospatial Thinking*, Lecture Notes in Geoinformation and Cartography, pages 221–236. Springer Berlin Heidelberg, 2010.

[104] W. Masri and Z. Mammeri. Middleware for wireless sensor networks: A comparative analysis. *Proceedings of the IFIP International Conference on Network and Parallel Computing Workshops*, pages 349–356, 2007.

[105] D. Massaguer, B. Hore, M. Diallo, S. Mehrotra, and N. Venkatasubramanian. Middleware for pervasive spaces: Balancing privacy and utility. In *Middleware 2009*, volume 5896 of *Lecture Notes in Computer Science*, pages 247–267. Springer Berlin / Heidelberg, 2009.

[106] Maté. `http://www.cs.berkeley.edu/ pal/mate-web`, 2002.

[107] B. W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451, 1975.

[108] M. Mitzenmacher. How useful is old information? *IEEE Transactions on Parallel and Distributed Systems*, 11(1):6–20, 2000.

[109] N. Mohamed and J. Al-Jaroodi. Service-oriented middleware approaches for wireless sensor networks. In *Proceedings of the 44th Hawaii International Conference on System Sciences*, HICSS '11, pages 1–9, Washington, DC, USA, 2011.

[110] M. M. Molla and S. I. Ahamed. A survey of middleware for sensor network and challenges. In *Proceedings of the 2006 International Conference Workshops on Parallel Processing*, pages 223–228, Washington, DC, USA, 2006. IEEE Computer Society.

[111] S. Mondal, R. Bhavna, R. Mohan Babu, and S. Ramakumar. Pseudo amino acid composition and multi-class support vector machines approach for conotoxin superfamily classification. *Journal of Theoretical Biology*, 243(2):252–260, 2006.

[112] D. Moodley and I. Simonis. A new architecture for the sensor web: The SWAP framework. In *Proceedings of the 5th International semantic web conference*, Athens, GA, USA, 2006.

[113] L. Mottola and G. P. Picco. Middleware for wireless sensor networks: an outlook. *Journal of Internet Services and Applications*, pages 1–9, 2011.

[114] A. Muñoz and J. Muruzábal. Self-organizing maps for outlier detection. *Neurocomputing*, 18(1–3):33–60, 1998.

[115] H. V. Nguyen, V. Gopalkrishnan, H. Liu, H. Motoda, R. Setiono, and Z. Zhao. Feature extraction for outlier detection in highdimensional spaces. In *Proceedings of the 4th Workshop on Feature Selection in Data Mining*, 2010.

[116] C. Noda, S. Prabh, M. Alves, C. A. Boano, and T. Voigt. Quantifying the channel quality for interference-aware wireless sensor networks. *ACM SIGBED Review*, 8(4):43–48, 2011.

[117] C. Noda, S. Prabh, M. Alves, T. Voigt, and C. A. Boano. *CRAWDAD data set cister/rssi (v. 2012-05-17)*. 2012. Published online at `http://crawdad.cs.dartmouth.edu/cister/rssi`.

[118] J. M. Norman. *Heuristic procedures in dynamic programming*. Manchester University Press, Manchester, 1972.

[119] A. Ollero, M. Bernard, M. La Civita, L. van Hoesel, P. J. Marron, J. Lepley, and E. de Andres. AWARE: platform for autonomous self-deploying and operation of wireless sensor-actuator networks cooperating with unmanned AeRial vehiclEs. In *Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics*, pages 1–6. IEEE, 2007.

[120] M. Onderwater. Detecting unusual user profiles with outlier detection techniques. M.Sc. thesis, `http://tinyurl.com/qf39up5`, 2010.

[121] OpenGeospatial-Consortium. `http://tinyurl.com/qy2ev22`, 2010.

[122] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572, 1901.

[123] S. Pedersen. *From calculus to analysis*. Springer, Cham, Switzerland, 2015.

[124] C. Phua, V. Lee, K. Smith, and R. Gayle. A comprehensive survey of data mining-based fraud detection research. Technical report, Monash University, 2005.

[125] R. Poli and J. Koza. Genetic programming. In *Search Methodologies*, pages 143–185. Springer US, 2014.

[126] S. Pollin, M. Ergen, S. Ergen, B. Bougard, L. Der Perre, I. Moerman, A. Bahai, P. Varaiya, and F. Catthoor. Performance analysis of slotted carrier sense IEEE 802.15.4 medium access layer. *IEEE Transactions on Wireless Communications*, 7(9):3359–3371, September 2008.

[127] D. M. W. Powers. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[128] E. Prud'Hommeaux and A. Seaborne. SPARQL query language for RDF. A W3C recommendation, available at `http://www.w3.org/TR/rdf-sparql-query/`, 2008.

[129] PULSENet. http://tinyurl.com/q2ufbpo, 2009.

[130] M. L. Puterman. *Markov Decision Processes: discrete stochastic dynamic programming*. Wiley, New York, NY, USA, 1st edition, 1994.

[131] V. Ravi and C. Pramodh. Non-linear principal component analysis-based hybrid classifiers: an application to bankruptcy prediction in banks. *International Journal of Information and Decision Sciences*, 2(1):50, 2010.

[132] P. Rawat, K. D. Singh, H. Chaouchi, and J. M. Bonnin. Wireless sensor networks: a survey on recent developments and potential synergies. *The Journal of Supercomputing*, 68(1):1–48, 2014.

[133] Y. Ren, V. Oleshchuk, F. Y. Li, and X. Ge. Security in mobile wireless sensor networks – a survey. *Journal of Communications*, 6(2):128–142, 2011.

[134] L. J. Rittle, V. Vasudevan, N. Narasimhan, and C. Jia. Muse: Middleware for using sensors effectively. In *Proceedings of the Second International Workshop on Networked Sensing Systems*, Mission Valley Marriott, San Diego, California, USA, 2005.

[135] SenseWeb. http://research.microsoft.com/en-us/projects/senseweb, 2007.

[136] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, volume 1, pages 134–141, 2003.

[137] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27(3):379–423, 1948.

[138] C. C. Shen, C. Srisathapornphat, and C. Jaikaeo. Sensor information networking architecture and applications. *IEEE Personal Communications*, 8(4):52–59, 2001.

[139] H. Shen and K. Chou. Using ensemble classifier to identify membrane protein types. *Amino Acids*, 32(4):483–488, 2007.

[140] K. Shi, Z. Deng, and X. Qin. TinyMQ: a content-based Publish/Subscribe middleware for wireless sensor networks. In *Proceedings of the Fifth International Conference on Sensor Technologies and Applications*, pages 12–17, 2011.

[141] A. Shyr, R. Urtasun, and M. I. Jordan. Sufficient dimension reduction for visual sequence classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3610–3617, 2010.

[142] D. Simon. *Evolutionary optimization algorithms*. Wiley, 2013.

[143] S. K. Smit and A. E. Eiben. Multi-problem parameter tuning using BONESA. In *Artificial Evolution*, pages 222–233, 2011.

[144] SNEE. `http://snee.cs.manchester.ac.uk/welcome.html`, 2008.

[145] E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal Ubiquitous Computing*, 10(1):37–44, 2005.

[146] F. M. Spieksma. *Geometrically ergodic Markov chains and the optimal control of queues*. PhD thesis, Rijksuniversiteit Leiden, Leiden, 1990.

[147] R. Sugihara and R. K. Gupta. Programming models for sensor networks: A survey. *ACM Transactions on Sensor Networks*, 4(2):1–29, 2008.

[148] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[149] B. G. Tabachnick and L. S. Fidell. *Using multivariate statistics*. Allyn and Bacon, Boston, 2001.

[150] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500): 2319–2323, 2000.

[151] TinyDB. `http://telegraph.cs.berkeley.edu/tinydb`, 2005.

[152] F. S. Tsai. Dimensionality reduction framework for blog mining and visualisation. *International Journal of Data Mining, Modelling and Management*, 4(3):267–285, 2012.

[153] P. UmaMaheswari and M. Rajaram. Principal component analysis-based frequent pattern evaluation on the object-relational data model of a cricket match database. *International Journal of Data Analysis Techniques and Strategies*, 1(4):364, 2009.

[154] M. F. Valstar, B. Jiang, M. Mehu, M. Pantic, and K. Scherer. The first facial expression recognition and analysis challenge. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition and Workshops*, pages 921–926, 2011.

[155] L. van der Maaten. Dimensionality reduction toolbox. Published online at `http://tinyurl.com/drtoolbox`, 2009.

[156] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[157] L. van der Maaten, E. Postma, and J. van den Herik. Dimensionality reduction: A comparative review. Technical Report 2009-005, Tilburg centre for Creative Computing, Tilburg university, 2009.

[158] M. M. Wang, J. N. Cao, J. Li, and S. K. Dasi. Middleware for wireless sensor networks: A survey. *Journal of Computer Science and Technology*, 23(3):305–326, 2008.

[159] D. J. White. Dynamic programming, Markov chains, and the method of successive approximations. *Journal of Mathematical Analysis and Applications*, 6:373–376, 1963.

[160] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *ACM Sigmod Record*, 31(3):9–18, 2002.

[161] A. Yener and C. Rose. Genetic algorithms applied to cellular call admission: local policies. *IEEE Transactions on Vehicular Technology*, 46(1): 72–79, 1997.

[162] L. Zhai, C. Li, and L. Sun. Research on the message-oriented middleware for wireless sensor networks. *Journal of Computers*, 6(5):1040–1046, 2011.

[163] Y. Zhang, N. Meratnia, and P. J. M. Havinga. *A taxonomy framework for unsupervised outlier detection techniques for multi-type data sets.* Number TR-CTIT-07-79. Centre for Telematics and Information Technology, University of Twente, Enschede, 2007.

[164] Y. Zhang, N. Meratnia, and P. Havinga. Outlier detection techniques for wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials*, 12(2):159–170, 2010.

[165] Y. Zhou, Y. Fang, and Y. Zhang. Securing wireless sensor networks: a survey. *IEEE Communications Surveys & Tutorials*, 10(3):6–28, 2008.

[166] Zigbee. `http://www.zigbee.org/`, 2007.

Summary

Networks of Sensors – Operation and Control

Over the past few years, the use of sensors has been growing at an unprecedented rate. Smartphones, intelligent washing machines, smart energy meters, and cars all work with a wide variety of sensors. On a larger scale, sensors are used for, e.g., patient observation in health care, smart building management, monitoring of infrastructural performance, and tracking wildlife. Further evidence of the popularity of sensors can be found on Gartner's 2015 Hype Cycle, which features 'The Internet of Things' – in which sensors play an important role – as one of the emerging technologies. With modern-day technology it is possible to make cheap 'mini-computers' that sensors can be attached to, yielding devices smaller than a credit card capable of monitoring of and interaction with the environment. In this thesis we refer to these devices as *sensor nodes*. Typically, besides sensors and basic processing capabilities, a node is also equipped with a radio that allows it to communicate wirelessly over short distances. Multiple nodes can form a network to jointly cover large geographical distances, i.e., form a *sensor network*.

Applications working with sensors and sensor networks face a number of challenges that are unique to sensor technology. Measurements from sensors are, for instance, subject to a certain amount of noise and can thus be unreliable. Also, sensor nodes are often battery-powered, and a nearly depleted battery might cause parts of a sensor network to be unreachable, or increase unreliability of measurements. Challenges such as these will become more relevant as the use of sensor technology gains in popularity in the near future. This raises the need for a deeper understanding of the challenges, for practical methods to deal with them, and for innovative solutions. This is the main motivation for the research in this thesis.

The chapters in thesis are grouped in two parts, with Part I consisting of three chapters on topics related to sensor networks and their challenges. Part II, also containing three chapters, deals with Markov Decision Processes (MDPs), a popular framework for controlling systems under uncertainty.

The first chapter in Part I is Chapter 2, and reviews middleware components for sensor networks. Typically, sensor technology uses a wide range of data formats and access protocols, and applications relying on sensor technology are forced to deal with this diversity. A middleware component forms a bridge between sensor applications and sensor technology, and hides the technological diversities from applications by offering a unified access point to the sensor technology. In Chapter 2 we review several types of component available in the literature, and then focus on a specific type that has gained in popularity recently. We call this a *centralized middleware component*, describe its general architectural form, and discuss four well-known centralized components. We finish with an outlook to future developments in the area of centralized middleware components.

In Chapter 3 we consider outliers (abnormal measurements) in sensor data, and how well dimensionality reduction techniques preserve these outliers. Dimensionality reduction is a family of techniques that aims to remove redundancy from data and create a shorter summary. Applying a dimensionality reduction technique to sensor data might reduce an outlier to a summary that is normal (compared to other summaries), thus losing the outlier and preventing applications relying on outliers to work with the short summaries. In Chapter 3 we describe three popular dimensionality reduction techniques, and experimentally determine how well they preserve outliers on a number of sensor data sets. The experiments identify one of the techniques as best able to preserve outliers, and we discuss the intuitions behind this result.

Chapter 4 deals with an important performance indicator of sensor networks: the *saturation throughput*. This property reflects how fast a sensor network can transmit measurements when many sensor nodes have a measurement to transmit. The network can only transmit one measurement at a time, so each node follows a set of rules to determine when it is allowed to transmit a measurement. This set of rules is described in the 'Media Access Control' protocol of IEEE 802.15.4, and instructs nodes to alternate a random waiting time with transmission attempts. Although the random waiting time allows multiple nodes to transmit measurements, it also causes the channel to be idle for short periods of time and thus to decrease throughput. In this chapter we provide a model for analyzing the saturation throughput of the IEEE 802.15.4 MAC protocol. Central to the model is the concept of a *natural layer*, which reflects

the time that a sensor node typically has to wait before sending a packet (as instructed by the MAC protocol). The key feature of the model is its simplicity compared to existing models in the literature. Also, it provides insight how the throughput depends on the protocol parameters and the number of nodes in the network. Validation experiments with simulations demonstrate that the model is highly accurate for a wide range of parameter settings of the MAC protocol, and applicable to both large and small networks.

The first chapter in Part II is Chapter 5, in which we consider the control of a queueing system. The controller of the system has to answer incoming queries with a response, which it can do by either forwarding incoming queries to the system (where it needs time for processing), or by responding with a previously generated response (incurring a penalty for not providing a fresh value). Hence, the controller faces a trade-off between data freshness and response times. Addressing the trade-off is traditionally done using a threshold policy. When the age of the database value exceeds a certain given threshold, fresh data is retrieved, and otherwise the latest database value is used. Although such policies are commonly used, there is room for improvement by setting a dynamic threshold: in cases where the information retrieval is time-consuming (as it is in wireless sensor networks), using a database value that is slightly above the threshold value might be acceptable. In Chapter 5 we model the system as an MDP, which turns out to be complex. In order to circumvent the complexities, we simplify the model and use this model to construct a control policy for the full, complex, model. Experiments with value iteration show that applying this policy leads to near-optimal performance and, in particular, that it performs significantly better than a traditional threshold policy.

In Chapter 6 we introduce *Value Function Discovery* (VFD), a novel method for discovery of relative value functions for MDPs. This method learns algebraic descriptions of relative value function by applying an Evolutionary Algorithm to sample points of the relative value function of an MDP. VFD's key feature is that the model parameters of the MDP are included in the discovered algebraic descriptions. The relative value function discovered by VFD can be used to, e.g., construct a policy for controlling a system with time-varying parameters. In Chapter 6, we describe VFD and apply it to an example MDP. We demonstrate that the discovered relative value function closely resembles the relative value function of the MDP. Additionally, we convert the discovered function to a policy, and demonstrate numerically that the resulting policy has excellent performance on a wide range of model parameters.

Finally, in Chapter 7 we continue work on VFD and again apply it to the example MDP of Chapter 6. This time, we include prior knowledge that the

optimal policy of the MDP is of threshold type. Instead of using MDP to discover a relative value function, we apply VFD to sample points of the threshold policy so that VFD discovers an algebraic expression for the threshold in terms of the model parameters. We demonstrate that this alternative use of VFD also yields near-optimal policies, illustrating that VFD is not restricted to learning relative value functions and can be applied more generally.

Samenvatting

Netwerken van Sensoren - Werking en Aansturing

De laatste jaren is het gebruik van sensoren met ongekende snelheid gegroeid. Smartphones, intelligente wasmachines, slimme (energie) meters, en auto's werken allemaal met een breed scala aan sensoren. Sensoren worden o.a. gebruikt voor het observeren van patiënten in een ziekenhuis, het beheer van klimaat in gebouwen, het monitoren van structurele eigenschappen van een brug, en voor het volgen van wild in een reservaat. De populariteit van sensoren blijkt ook uit de 2015 editie van de Gartner Hype Cycle, waar 'The Internet of Things' – een onderwerp waar sensoren veelvuldig gebruikt worden – genoemd wordt als één van de opkomende technologieën. Dankzij recente technologische ontwikkelingen is het mogelijk om sensoren te koppelen aan goedkope 'mini-computers', resulterend in apparaten die kleiner zijn dan een credit card en toch voldoende rekenkracht hebben om simpele applicaties uit te voeren. Behalve sensoren bevatten deze apparaten meestal ook een draadloze verbinding die geschikt is voor het verzenden van data over korte afstanden. Om grotere afstanden te overbruggen vormen de apparaten gezamenlijk een netwerk, zodat toepassingen in een groter geografisch gebied ook tot de mogelijkheden behoren.

Toepassingen die gebruik maken van sensoren of sensornetwerken worden geconfronteerd met een aantal uitdagingen die uniek zijn voor sensortechnologie. Metingen van sensoren hebben bijvoorbeeld meestal een bepaalde onnauwkeurigheid. Verder werken sensoren vaak op een batterij, en een lege batterij kan ervoor zorgen dat een deel van het netwerk onbereikbaar wordt. Zulke uitdagingen worden steeds belangrijker naarmate het gebruik van sensoren blijft groeien in de nabije toekomst. Een grondig begrip van deze uitdagingen, goede methodes om er mee om te gaan, en innovatieve oplossingen zijn hierbij noodzakelijk. Dit is de belangrijkste motivatie voor het onderzoek in dit proefschrift.

Het proefschrift bestaat uit twee delen, waarvan het eerste deel drie hoofdstukken bevat rond het thema 'sensornetwerken'. Het tweede deel bevat drie hoofdstukken rond het onderwerp 'Markov-beslismodellen', een populair raamwerk voor het modelleren van beslisproblemen in onzekere omstandigheden (welke vaak voorkomen in sensornetwerken).

Het eerste deel start met een behandeling van middlewarecomponenten voor sensornetwerken in Hoofdstuk 2. Sensoren en sensornetwerken gebruiken protocollen en gegevensformaten die vaak verschillen per fabrikant, zodat applicaties genoodzaakt zijn om met deze diversiteit om te gaan. Een middlewarecomponent vormt een brug tussen (netwerken van) sensoren en de applicaties, en verbergt de diversiteit aan formaten achter een uniforme interface die bruikbaar is voor alle applicaties. Hoofdstuk 2 behandelt een aantal types componenten, en richt zich uiteindelijk op de recentelijk populaire 'centrale middlewarecomponenten'. De algemene architectonische kenmerken van dit type worden besproken, gevolgd door een beschrijving van vier verschillende voorbeeldcomponenten. Het hoofdstuk eindigt met een vooruitblik naar toekomstige ontwikkelingen op het gebied van centrale middlewarecomponenten.

In hoofdstuk 3 richten we onze aandacht op uitbijters (abnormale metingen in sensordata), en wordt onderzocht hoe goed zogenaamde dimensiereductietechnieken in staat zijn deze uitbijters te behouden. Een dimensiereductietechniek is gericht op het verwijderen van redundantie uit data, en vormt een soort samenvatting van de originele data. Het toepassen van een dimensiereductietechniek kan er toe leiden dat een uitbijter in de originele data vervolgens tussen de samenvattingen geen uitbijter meer is. Applicaties die afhankelijk zijn van uitbijters (bijvoorbeeld een toepassing die een alarm stuurt bij te hoge CO_2 metingen) kunnen dan geen gebruik maken van de korte samenvattingen. De toenemende hoeveelheid sensordata betekent echter dat het gebruik van de korte samenvattingen wel degelijk wenselijk is voor deze applicaties is vanuit een rekenkundig oogpunt. Hoofdstuk 3 behandelt drie populaire dimensiereductietechnieken, en onderzoekt aan de hand van een aantal kwaliteitscriteria hoe goed de drie technieken in staat zijn om uitbijters te behouden. De experimenten laten zien dat één van de drie technieken het beste uitbijters behoudt, en we bespreken de intuïties achter dit resultaat.

Hoofdstuk 4 gaat over een belangrijke prestatiemaat van een sensornetwerk: de *doorvoercapaciteit*. Deze maat geeft aan hoe snel een sensornetwerk in staat is om een meting te versturen in een situatie waar vele andere sensoren in het netwerk dat eveneens willen doen. Het draadloze kanaal dat het netwerk gebruikt is slechts in staat om één meting tegelijkertijd te versturen, dus iedere sensor volgt een stelsel van regels om te bepalen wanneer deze een meting mag

versturen. De regels zijn beschreven in het 'Media Access Control' protocol van de IEEE 802.15.4 standaard, en zorgen ervoor dat sensoren afwisselend een bepaalde hoeveel tijd wachten en een verzendpoging doen. Het wachten resulteert erin dat meerdere sensoren metingen kunnen versturen, maar zorgt er tegelijkertijd ook voor dat het kanaal nu en dan niet gebruikt wordt, wat weer nadelig is voor de doorvoercapaciteit. In dit hoofdstuk presenteren we een model voor het analyseren van de doorvoercapaciteit van het IEEE 802.15.4 MAC protocol. Het model draait om het concept van de *natuurlijke laag*, welke correspondeert met de tijd dat een sensor normaal gesproken moet wachten (volgens het MAC protocol) voor het verzenden van een meting. Vergeleken met bestaande modellen in de literatuur onderscheidt ons model zich door zijn eenvoud. Het model biedt inzicht in hoe de doorvoercapaciteit afhangt van de protocolparameters, en van het aantal sensoren in het netwerk. Experimenten met simulaties laten zien dat het model zeer nauwkeurig is voor een brede selectie van waarden voor de protocolparameters, en dat het toepasbaar is op zowel grote als kleine sensornetwerken.

Deel twee van het proefschrift, over Markov-beslismodellen, start met Hoofdstuk 5 en bekijkt de aansturing van een wachtrijsysteem. De controller van het systeem ontvangt aanvragen voor metingen uit het wachtrijsysteem, en kan deze beantwoorden door een nieuwe meting op te vragen bij het systeem, of door een eerder gedane meting te gebruiken als antwoord. De eerste optie (een nieuwe meting ophalen) kost echter relatief veel tijd, zeker bij grote drukte in het systeem. De tweede optie (een eerder gedane meting teruggeven) kost geen tijd, maar geeft een meting terug die al wat ouder is. De controller maakt voor zijn beslissing dus een afweging tussen de tijd die nodig is om een nieuwe meting op te vragen bij het wachtrijsysteem, en de leeftijd van de eerder gedane meting. In de praktijk wordt hiervoor door de controller vaak een drempelwaardestrategie gebruikt, waarbij het wachtrijsysteem gebruikt wordt als de leeftijd van de eerdere meting boven een bepaalde drempelwaarde komt. Ondanks de populariteit van drempelwaardestrategieën is er ruimte voor verbetering, zeker wanneer het systeem erg druk bezet is. In dit scenario kan het de moeite waard om soms toch een wat oudere waarde terug te geven wanneer het opvragen van een nieuwe meting teveel tijd in beslag zou nemen. In Hoofdstuk 5 modelleren we het beslissingsprobleem van de controller als een Markov-beslismodel. De complexiteit van het model verhindert een exacte analyse, maar met een versimpeling van het model zijn we toch in staat om een beslissingsstrategie voor de controller af te leiden. We laten met experimenten zien dat deze beslissingsstrategie aanzienlijk beter presteert dan de drempelwaardestrategie, en dat het meenemen van de drukte van het wachtrijsysteem in het beslissingscriterium dus de moeite waard is.

In Hoofdstuk 6 introduceren we *Value Function Discovery* (VFD), een innovatieve methode voor het vinden van relatieve waardefuncties van Markov-beslismodellen. Deze methode leert algebraïsche uitdrukkingen van relatieve waardefuncties door het toepassen van een Evolutionair Algoritme op numerieke benaderingen van deze waardefuncties. Het belangrijkste aspect van VFD is dat de parameters van het Markov-beslismodel inbegrepen zijn in de algebraïsche uitdrukkingen die VFD vindt. De gevonden relatieve waardefuncties kunnen bijvoorbeeld gebruikt worden voor het opstellen van een beslisstrategie voor de aansturing van een model met tijdsafhankelijke parameters. Hoofdstuk 6 beschrijft VFD en past de methode toe op een voorbeeld Markov-beslismodel. We laten door middel van experimenten zien dat VFD een waardefunctie vindt die de waardefunctie van de optimale strategie dicht benadert, en dat de prestatie van de corresponderende beslissingsstrategie zeer goed is vergeleken met de optimale strategie.

Het laatste hoofdstuk van dit proefschrift, Hoofdstuk 7, gaat eveneens over VFD en past de techniek wederom toe op het Markov-beslismodel uit Hoofdstuk 6. Echter, dit keer gebruiken we voorkennis dat de optimale strategie voor het beslismodel een drempelwaardestrategie is. We passen VFD op een slimme manier toe, zodat deze een algebraïsche uitdrukking vindt voor de drempelwaarde (in plaats van de relatieve waardefunctie) in termen van de modelparameters. Resultaten van numerieke experimenten laten zien dat deze alternatieve toepassing van VFD ook resulteert in een beslissingsstrategie met uitstekende prestaties, en dat VFD dus niet beperkt is tot het leren van relatieve waardefuncties.