Optimization of Information Acquisition for Decision-Intensive Processes

This thesis is part of the PhD thesis series of the Beta Research School for Operations Management and Logistics (onderzoeksschool-beta.nl) in which the following universities cooperate: Eindhoven University of Technology, Ghent University, Maastricht University, University of Twente, VU Amsterdam, Wageningen University and Technology, CWI, UHasselt, Tilburg University, University of Antwerp, VU Brussels and KU Leuven.

This work is part of the research programme Real-time data-driven maintenance logistics with project number 628.009.012, which is supported by the Dutch Research Council (NWO).

A catalogue record is available from the Eindhoven University of Technology Library.

ISBN: 978-94-6469-299-0

Cover: Designed by Wendy Schoneveld

Printed by: proefschriftmaken.nl

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Beta Research School for Operations Management and Logistics

Optimization of Information Acquisition for Decision-Intensive Processes

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. F.P.T. Baaijens, voor een
commissie aangewezen door het College voor
Promoties, in het openbaar te verdedigen
op 9 Mei 2023 om 16:00 uur

door

Simon Voorberg

geboren te Sliedrecht

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Mathematics is the language with which God has written the universe.

Galileo Galilei

Contents

1 Introduction

Decisions are one of the key factors in everyday life. We are often unaware of the amount of decisions that we make but estimations go up to 35,000 decisions per day for an average adult [Sahakian and LaBuzetta, 2013]. This number is very likely to only increase in the near future since the amount of available information is also increasing. The modern society praises itself for the endless options that can be offered for almost any product to every individual's taste. What is however less discussed is the consequence that comes up from all of this, which is the enormous complexity that humans have to act upon.

A decision in general requires the weighing of multiple factors that affect the outcome of a decision. We define such factors henceforth as the attributes or *information attributes* of the process. The attributes impact the individual both before the decision, as the value of the attributes needs to be judged, and after, where the individual can experience the outcome or response of the decision. The available attribute values are crucial to assess while at the same time the process of obtaining information attributes is time-consuming. Hence, it is complex to determine what information is the most valuable to have, to improve the quality of the decision. The result is an information-acquisition process where decisions are made on what information to collect, concluding in a *final decision* that should improve when more information is available. Many of the daily decisions happen in such haste that this process of collecting information is almost automatic, but there is a small amount of everyday decisions that require the serious consideration of the pros and cons such that the best option is chosen. An example of a complex decision is buying a house. In the case of buying a house it is important to have an idea of what neighbourhood the house is in, distances to important locations and the state of the house. Many of these information attributes are not directly known to the buyer. Moreover, which of those attributes is the most crucial depends both on the valuation of the decision maker and the attribute value or outcome. It is therefore hard to tell if one needs to invest more time in experiencing the neighbourhood or in determining the state of the house.

Businesses suffer from the same problem, where the amount of decision alternatives, the amount of information attributes, and the complexity of the decision process are all playing a role. Especially in this era of industry 4.0, where information is key, human decision makers will be less and less comfortable in keeping oversight of such complex processes. In this thesis, we will focus specifically on the optimization of a decision process that changes dynamically due to new information inputs, often indicated as a *decision-intensive process* (DIP).

Bromberg [2007] and Vaculín et al. [2011] were the first to come up with a definition for decision-intensive processes. We define decision-intensive processes as *Complex repeatable business processes that require a knowledge worker to decide on performing a number of tasks to decrease the uncertainty of the information position until a concluding decision can be taken that maximizes an organizational objective. When specifically using knowledge from a human or algorithm to optimize the process, these processes are also called knowledge-intensive processes*. From this definition we can derive multiple important characteristics that are crucial to consider when optimizing a DIP. First, the process is "complex and repeatable". This indicates that we have to find a method that takes into account that the process is performed many times with each time different information input resulting in different decisions. Second, knowledge workers are required to untangle the web of complex impact that the information has on the outcomes of the decisions. This is done by performing so-called *tasks*. These tasks can be anything as long as they eventually improve the *information position*, i.e., the set of available information attribute values, for the decision that should be made. Especially this second necessity is something that we try to optimize by assisting the knowledge worker in making correct decisions. DIPs can reach levels of complexity due to the amount of tasks and information attributes that impact each other and cannot always be well understood by human intelligence anymore. That is why we propose to introduce a specific idea of modeling of DIPs that helps in optimizing the decisions that appear in a DIP. This modeling consists of two components: 1) An approach that helps to turn DIPs into optimization problems and 2) an optimization algorithm that turn this optimization problem into a solution/policy for the DIP.

A specific field of research where decisions are made continuously and also have to be adapted almost continuously is maintenance optimization. Companies with high-value assets need to use those assets in production while also keeping their assets in excellent condition. In the airline industry it is rather common to use an inventory of spare parts such that the asset can always continue running while specific components of the airplane are revised. These revisions often are performed by specialized companies that have long-running maintenance contracts. Especially when closing such contracts, decisions are crucial as they impact the reward on the contract for several years. Therefore, there is often an official tender procedure in which multiple companies can bid for getting the contract. The maintenance provider has the difficult job to collect as much information as possible to be able to set a profitable price for the tender while also beating the competitors in the market. The DIP consists in this example of a set of information-acquiring tasks which can be executed to collect information about the tender and one decision-making task on offering a tendered price based on the collected information.

While this thesis starts with the modeling of a DIP such that we can understand the effect of the process steps on the decision, we require a second step. This is the step where we convert the DIP model into an optimal policy that is obtained by a mathematical model. The field of Operations Management (OM) focuses on how to model problems and solve these models to find the best solution or decision. In the case of finding an optimal DIP policy, we introduce a model with the collection of input parameters (attributes) as random variables, which can be converted into single attribute values by retrieving the attribute value. That means that we have to optimize a combination of three subdecisions:

- Which attribute values to collect and in what order to collect?
- When to stop collecting information attributes?
- What final decision to take based on the attribute values that have been collected?

Coming back to the example of buying a house, the three subdecisions that one needs to decide on while in the buying process are 1) which information to collect about the house that is the most important to know while considering to buy the house, 2) when to stop collecting information and continue to the final decision, and 3) what final decision to make. Furthermore, the DIP model can infer different constraints on the optimization problem. There can be, for example, tasks that are necessary to be able to make a final decision or there can be tasks which can only be performed after another task has been performed. Such constraints must also be translated correctly to the optimization model. For buying a house this can translate in the necessity to have an approved mortgage sum that can pay for the house or the possibility to have a viewing of the house only after approval by the current owners.

Although we introduce techniques in this thesis to find optimal policies, we confirm the ongoing need for human decisions makers. Our policies are optimized for the problem based on expected values, i.e., over many cases there is a profit improvement. However, a human decision maker can always judge every individual case and realise what is the reason that sometimes a counter-intuitive decision must be taken. That is why in multiple places in this thesis we will stress that a human decision maker is helped with the decision support during a DIP and how we make the solution explainable and useful for the decision maker.

In this chapter we will provide an introduction to DIPs both from an Information Systems perspective (Section 1.1) and from an Operations Management perspective (Section 1.2). Section 1.3 introduces the research objective and the used methodologies to do so. In Section 1.4 we will introduce the different contributions of this thesis. Finally, Section 1.5 will give an outline of the remainder of this thesis.

1.1. DIPs from an Information Systems perspective

The research field of Informations Systems has a dedicated subfield that focuses on the modeling and management of processes: Business Process Management (BPM) [Dumas et al., 2013, Jeston, 2014]. Business processes are seen here as a chain of unpredictable event arrivals, e.g. *the arrival of information attribute values*, reactive activities, e.g., *human tasks*, and calculated decisions. A process that has completely been defined in steps and where the amount of unexpected events is limited is referred to as *structured*. A process where every instance is different and consists of only unpredictable behaviour is *unstructured*. Finally, processes, such as DIPs, where there is a possibility of modeling a part of the process only, but where still unpredictable events happen are referred to as *semi-structured*.

BPM started with a particular interest in workflow processes which have a structured process flow, but lately there is a clear shift happening to a more holistic view consisting of multiple types of processes. Since *work flow* processes become more and more automated, the focus is changing also to *information* processes and subsequently also *decision-making* processes. What is clear though, is that the three different kinds of process streams constantly interact with each other in a DIP environment. Only looking at one of the three process models would give the idea of an unstructured process, while the combined view is more semi-structured. A DIP can involve procedural steps that are necessary and, similarly, a DIP requires a lot of information, making the process of collecting information but also of using information extremely important. A separate process model focusing on workflow or information would not suffice in this case. That is why we impose a combined view of process models that interact. In the next three paragraphs we will introduce the three main process components that are used in BPM and explain their independent needs but also why they are all interconnected.

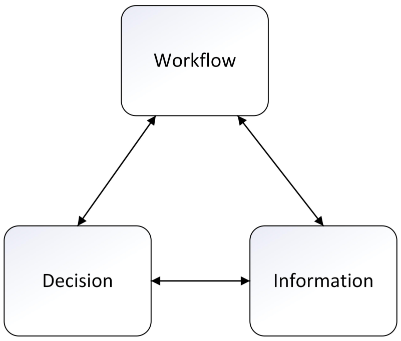


Figure 1.1: Holistic view on Business Process

Decision The most important component for a DIP is the *decision*. Decisions impact processes. The outcome of a decision is often determining what actions in the process are taken next. In the field of BPM, decisions can be modeled in many ways but we focus on the Decision Modeling Notation (DMN) [Object Management Group, 2020]. This notation uses what is called a Decision Requirement Graph (DRG) to completely define a decision in a Business Process. A DRG shows what is required to make a decision. This can consist of information attributes but can also be a decision authority (someone who is empowered to make the decision) or even a knowledge model (a model that defines how the information attributes are affecting the decision). Note that knowledge models can be expressed in different ways by, for example, a decision table or a specific literal expression. It is important to mention that a decision outcome can function as an input parameter for another decision. All the interactions between different decisions that happen in one DIP can therefore be modeled in a single DRG. There is however one problem to this language that makes it complicated to use for optimization of the full process. DMN standards assume that all information should be available before one can proceed to the decision making step. In the case of optimizing a DIP, this assumption cannot be followed, since obtaining the information is part of the process. Hence, DMN is not suited to model a DIP in such a way that we can derive a correct optimization model out of it.

Workflow The *flow* of events or procedures is part of a DIP. In Business Process Management the most recognized notation used for this is Business Process Model and Notation (BPMN) [Object Management Group, 2010]. BPMN assumes, apart from some subroutes, always a structured flow of events, opposite to Case Management Modeling Notation (CMMN) which allows for semi-structured process modeling and is introduced in the next paragraph. BPMN is used to describe a predictable process flow. One can clarify who is involved in which event, and can also store and retrieve data in a BPMN model or set constraints based on information, but information is not the leading variable in the process. Since information is the determining factor in DIPs (information leads to better decisions), we use CMMN models.

Information In the previous paragraphs we already discussed the need for *information attributes* to make a decision. Information is the crucial parameter that impacts the flow of the process and also impacts the value of the decision. Hence, besides defining how these attributes impact the decision, they should also be connected to a task which defines the underlying process of collecting/acquiring that information.

That is why there is the Case Management Modeling Notation [Object Management Group, 2015]. CMMN consists of a case plan and a case file. The case file is often denoted using existing languages such as UML, whereas the case plan is created using special notation. This notation allows to build cases that consist of tasks. These tasks are not necessarily connected in a procedural way: there is no pre-defined flow to follow. Instead, the data, the rules/constraints and the authority, often a decision maker, together determine which path of tasks is taken. A task can have an entry criterion and an exit criterion depending on data that determines when the task can be started and when the task is finished. Finally, tasks can be grouped in stages. In conclusion, CMMN is a language that perfectly defines what the impact is of information on the progress of a business process. That is why for a DIP, CMMN is excellent to use. The information that is used in a CMMN model should also have a structure. For this, existing models such as class diagrams, ER diagrams or UML diagrams can be used.

In the field of Business Process Management everyone is aware of the value of being able to model any kind of business process even considering the interplay between decisions, information and process and any level of available structure in the process. However, what is missing for the current languages in BPM is an approach to use them to model a DIP such that it can be **optimized**. In this thesis we will introduce approaches that use the above introduced modeling languages of CMMN and DMN to optimize DIPs both by defining how to convert a DIP model in an optimization problem and by introducing/studying methods that can optimize these problems. This optimization is achieved by introducing additional constraints or constructs to the current methods of business process modeling, allowing to apply those languages in a specific way.

1.2. DIPs from an Operations Management perspective

While in this thesis we start conceptually from modeling a DIP using BPM methods, the ultimate goal is to optimize these processes. The general goal of the field of Operations Management is to model operations, optimize decisions and implement solutions. Based on a (large) number of input parameters, the aim is to maximize an objective function that depends both on a decision and these input parameters. Finding the best decision results in the highest or lowest objective, often related to the minimization of cost or time or the maximization of profit. Such decisions appear daily in the operations environment of a company. However, the variability in possible decision variables, optimization objectives and input parameters causes the need for a whole set of optimization methods. In the case of a DIP there are some very clear indicators for the kind of problem. When the input parameters (information attributes) are uncertain, we use probabilistic models that optimize the expected output based on the probability distributions. Furthermore, if there are sequential decisions that have to be taken we use dynamic programming methods that optimize the long term profit taking into account all future actions. Together, these two modeling assumptions lead us in the direction of so-called Markov Decision Processes [Puterman, 2014].

A general assumption that often exists in the optimization of a problem is the availability of the parameters. That is, either the assumption is that the exact value of the parameter is known or the assumption is that the probability distribution of the parameter is known and can be used to optimize. In the case of DIPs, however, we create a more rare situation where the attribute starts as a random variable with a known probability distribution but can turn into a known value after paying a certain amount of money. What happens therefore is that every phase where a new information attribute reveals its true value, we have to solve a new problem since the set of input parameters has changed and hence the outcome of the objective function has changed. These continuous changes clearly require a dynamic method to solve the complete problem of optimizing the process.

A Markov Decision Process (MDP) is used to solve problems based on four important constructs that have to be defined: state, actions, reward, transitions. The state defines the current status of the process. It contains all the parameters that can change in the course of the process, i.e., in the case of a DIP the state consists of the known attribute values, the yet unknown attributes, and possibly a chosen decision alternative. Each decision epoch, the decision maker should decide on what to do by taking an action. This action can be to collect more information by acquiring an attribute value or it can be to choose a final decision. Furthermore, each time step a cost can be incurred being the acquisition costs or a reward can be obtained by making a final decision. All costs together in the course of the process and the reward lead to a profit that defines the efficiency of the policy. Finally, transitions make the process go from one state to another depending on the chosen action. In this case, the transitions are based on uncertainty and can therefore be expressed in transition probabilities. Note that these processes are finite since the process is concluded immediately after the final decision has been taken.

The final reward function as a consequence of the final decision is defined based on the true value of the attributes together with the final decision. This final decision sets a value for the decision variable that we try to optimize in the end. Hence, this function is defined as a deterministic function. Because of the uncertainty of the input parameters, the function can not be used deterministic. We cannot use a normal dynamic program but we use a Markov Decision Process, which assumes that the next state only depends on the information of the current state and transitions based on the probability distributions of the information attributes.

1.3. Research Objective and Methodology

The field of decision optimization has seen many approaches already. What is new to our approach however is the combination of Business Process Management and Operations Management, specifically toward DIPs. We use the knowledge in the field of BPM to model DIPs in the best way by taking into account all three main components of the process: knowledge/decision, information and process. By defining a final goal that should be optimized based on the decision and by deriving any additional constraints that appear due to the information or process view, we aim at modeling the full process including all three main components. However, that is not where our approach stops. Having the process in a model is one step, but the second step is to make sure this model, hence the process, is efficient. Therefore, we introduce an optimization method that can find policies, in some cases optimal, such that the DIP can be handled as efficient as possible, while still making the right decisions.

Main Goal: Modeling and optimization of decision-intensive processes.

In our solution we will take into account the different uncertain outcomes of information attributes. By modeling the information attributes as random variables, we take into account their uncertainty while not sailing blindly. Simply by determining the probability distributions and including that information when optimizing the process, we can derive in expectation an optimal policy. Referring also to the first of two points that we made about the DIP definition by Bromberg [2007], the high repetitiveness allows us here to use this approach of optimizing in expectation.

In the first four main chapters we look at the problem of modeling and optimizing DIPs. Chapter 2 introduces a basic approach to transform a Business Process Model into an optimization problem for a specific set of DIPs for which we show the effectiveness of the approach on a case study, including an online demonstrative tool. Chapter 3 continues on this approach making use of the natural hierarchy in companies to decompose large DIPs into smaller problems that can be solved leading to policies for DIPs that are in total intractable. Chapter 4 focuses on the optimization part of the approach where we compare the optimal policy to some heuristics that simplify the behaviour of the policy, which makes it also easier to use for human decision makers. Moreover, we introduce a case study inspired by the Quotation Optimization process of Fokker Services. Finally, in Chapter 5 we introduce throughput time constraints, also allowing parallel information collection. This helps the decision maker in improving the decision strategy and also allows to follow more naturally the use of process modelling according to BPM.

Chapter 6 will focus on a more specific topic where we optimize the traveling policy of a maintenance engineer in a network of assets. This can be seen as a typical example of a DIP, although the constraints cause the MDP solution to also be slightly more complex. The problem plays in the same applied area of maintenance optimization but is based on a practical problem at Philips. We show the value of using reinforcement learning as a solution for the problem and derive some interesting managerial insights.

1.4. Contributions of this thesis

This thesis contains five core chapters. The first four core chapters focus on optimizing DIPs, while the fifth core chapter solves a Dynamic Traveling Maintainer Problem. In this section we briefly introduce the five chapters and describe the contributions we have made to the literature in both Business Process Management and Operations Management. We also present the managerial implications that follow from these contributions.

1.4.1 Optimizing information gathering in Decision-Intensive Processes

Chapter 2, which is based on Voorberg et al. [2021], introduces a new approach that can be used to model a DIP based on CMMN modeling language and subsequently derives a Markov Decision Process from this DIP model. Consequently, we introduce the notation and the framework to optimize this MDP and find the optimal policy.

This chapter is the first step in solving the problem as introduced in the introduction of this chapter. We start from a DIP and argue what is the correct business process approach to model such processes. We show that there is currently a lack of approaches that specifically target the optimization of DIPs. Based on a maintenance quotation process as motivating example, we show how a DIP can be modeled using the CMMN modeling language. To be able to model DIPs using CMMN, we impose some assumptions on the DIP, which we define as the subclass of Optimizable DIPs (ODIP). An ODIP is assumed to always directly collect the right information, i.e., information is stable. Furthermore, the timing of information collection is not taken into account. Based on these assumptions we aim to reach the goal of: *Defining an approach that guides decision makers in making a decision in DIPs in the most efficient and effective way.* We also introduce the notion of an Information Structure which represents a function that structures all the available information and defines how they impact the final decision. This can be, for example, a decision tree. The Information Structure is used as the objective function that is optimized in the MDP.

The result of the paper is an approach that consists of five main steps after which we define how the outcome of this approach can be deployed. The five steps are:

1. Create CMMN model
2. Retrieve MDP structure
3. Filter MDP
4. Create Information Structure
5. Solve MDP

We apply the introduced approach to a motivating example that comes from a real aircraft component maintenance company (Fokker services) to show the plausibility of it and use a demonstrator to show the effectiveness of the approach. Using a decision tree that reflects the current behaviour of the decision maker at the company, we make a comparison to show the performance and profit improvement of the new approach. We can show that in some cases it is possible to double the profit resulting from the process. Also, using the demonstrator we can show the effect of the approach in giving decision support, meaning that the optimization policy adapts to opposing decisions of the decision maker by giving a new advice based on the new situation. Based on this we conclude that it is possible to support human decision makers in optimizing how to run a DIP.

1.4.2 Hierarchical decision making in Decision-Intensive Processes

In Chapter 3 we introduce a new approach building on Chapter 2 that makes use of the natural hierarchy that often exists in decision making environments to decompose complex DIPs into smaller size problems that can be optimized. We make use of the ODIP approach but adapt it such that it can contain a multitude of decision points in one DIP. To allow for this we require the use of a Decision Requirements Graph (DRG) from the DMN standard [Object Management Group, 2020] that defines the interconnection of data attributes and subdecisions on different levels.

The main motivation for this chapter results from two things. First, assuming that subdecisions always result in a final decision, one can always merge a tree of multiple subdecision processes into one (final) decision process. This DIP, however, is very quickly intractable to solve. Hence, we can use the decomposition to decrease the size of a complex DIP allowing it to become tractable and solvable again by our MDP solution. Second, if we zoom in on how DIPs are often organized in companies, we see that there are multiple departments involved in the same DIP. Each department is asked to give their own input, often consisting of their own subdecision. Hence, using this natural decomposition is not only allowing us to solve larger problems but also comes closer to reality. The goal is: *Use the natural hierarchy in DIPs to decompose the optimization problem in solvable subproblems.*

A DRG consists of a set of hierarchical subdecisions that are input to each other. Information and knowledge are input that define how a subdecision can be taken. Each subdecision in the DRG can be modeled as an independent ODIP because the outcome of the ODIP is in itself again a data attribute that is a random variable and can map as input information to a higher subdecision. The complex DIP is optimized by solving each ODIP in a bottom up approach of the decision hierarchy.

Using this new approach of decomposed optimization, we aim to find policies that perform well. Though we know that the result of our solution is suboptimal, we show for small examples where the complete DIP is still solvable how well our solution performs. Furthermore, we introduce a case study for a bicycle production company for which we show that it is very well possible to decompose their DIP into smaller subprocesses. We conclude that for complex DIPs that often contain multiple involved departments in a company, the new approach is a useful way of modeling and optimizing.

1.4.3 Information Acquisition for Service Contract Quotations by Repair Shops

In Chapter 4, based on Voorberg et al. [2023], we introduce a model for optimal dynamic information acquisition for profit maximization (DIA model) and show how effective it can be to use this model in an industrial environment to do service contract quotations for repair shops. The process of information acquisition is a strong example of a decision process where information is crucial to make the right decision. Although in Chapter 2 we already show the value of modeling and optimizing DIPs, in this chapter we focus more on the application of the model in a particular environment and compare the optimal policy to possible other policies that can simplify the process and make the interpretation for decision makers easier. The motivation for this comes from a collaboration with Fokker Services where we realised that a very complex policy can be more confusing for decision makers, making it harder for them to accept and adopt this new policy. If we are able to find more easily comprehensible policies that still remain effective, we could sacrifice some profit while having the full support of the executing human decision makers.

Based on the DIA model we introduce a refinement of this model that focuses on quotation optimization (QO model). We introduce which attributes play a vital role in making the right decision on what price to quote. We also discuss in further detail how those attributes connect to each other and together are part of the objective function or information structure. Using the QO model, we introduce a case study example, found at Fokker services. For this case study example we show in a full factorial experiment what is the sensitivity to different modeling assumptions and we also show how time and cost can be the variables in a trade-off that lead to an efficient frontier. The goal of this chapter is: *Find the best policy when we optimize a quotation process using the Dynamic Information Acquisition model.*

In the analysis we do not only look at the optimal solution of our model, but also make a comparison with some heuristics that take away some of the complexity of the decision. In the introduction of this chapter we described three key subdecisions that together are the heart of a DIP. By removing one or more of these subdecisions in the dynamic optimization and replacing them by a pre-determined policy we can simplify the process both in terms of computation time but more importantly in terms of comprehensibility. We show that fixing the order of the attributes has a small effect on the reward of the process, while completely fixing the set of attributes that is revealed has a much larger effect. Furthermore, we show the efficient frontier for these different heuristics compared to the optimal policy when constraining the average invested time.

1.4.4 Scalable algorithms for throughput time constrained complex decision making processes

In Chapter 5 we introduce a method that extends on Chapter 4. This is necessary because we are extending also the model that we introduced in Chapter 4. Next to process costs that minimize the acquisition of information, we introduce a throughput time constraint, i.e., we do not allow for the process to take longer than a given time threshold. The new model is a time-constrained DIA model. However, to include this, we have to adapt our model of Chapter 4. First, we have to allow tasks to be chosen in parallel. The time threshold does not allow for a solution where every attribute is collected sequentially. Hence, to still allow for all information to be collected we need tasks to happen in parallel. Parallel tasks allow to collect more information in a limited amount of time, while complicating the possible policies that can be followed. Consequently, each task's individual timing needs to be tracked since there is no pure sequential policy anymore.

The motivation why we introduce this model extension is based on the following argument. In Business Process Management it is common to allow for tasks to happen in parallel. A clear example why tasks can be modeled in parallel is that enterprises often make use of multiple agents and departments for a single decision process. In combination with the pressure of deadlines, that means that these agents can do tasks at the same time, making it necessary for the model to include parallel tasks. The result of the extensions is that we can introduce a throughput time constraint. In an environment such as quotations, this is a very common rule. Hence, the extension will further improve the policy that can be used in such environments. The goal of this chapter is: *Build a smart algorithm to solve a Time-constrained Information Acquisition problem*.

Since the problems that were introduced in Chapter 4 could not be too large due to computational limits, it would also be preferred to find a better technique for finding the best policy. Even more, when adding variables to the problem that keep track of the attribute's individual times, we should use a method that can handle state spaces of significantly large size. That is why we make use of deep reinforcement learning. The method uses reinforcement learning, i.e., simulation of the model, to improve the policy and it uses a neural network to learn a correct policy from only a small subset of all the possible states that exist. Opposite to the exact method of using backward recursion in Chapter 4, we use a DRL method and show how effective it is in finding a good policy. First, by comparing it to the optimal solution in Chapter 4 for a small size problem. Second, we show how well it performs compared to some of the basic greedy policies that can be applied.

1.4.5 Policies for the Dynamic Traveling Maintainer Problem with Alerts

In Chapter 6, based on Da Costa et al. [2023], we introduce the Dynamic Traveling Maintainer Problem with Alerts where an engineer is traveling around in a network of assets and is tasked with the maintenance to keep all assets working. The distance between the assets, the costs of repairing the assets and the lost reward due to unavailable machines are combined optimized in this problem. This complex problem can be seen as a DIP. However, being able to incur all the available information to find an optimal traveling policy requires a rather specific solution method. We introduce three solution directions for which we show the effectiveness in different circumstances.

In this chapter, we focus on a new kind of problem which stems from the Traveling Maintainer Problem. However, Philips, the company that inspired us for this chapter, made us realise that this is not a static problem that can be solved once every few weeks or days to find a an optimal routing for the engineer. A static solution to this problem would mean that given the current situation, a complete routing plan is determined for the maintainer for the coming days. However, failures from machines can happen at any time and based on them the situation can change so much that the routing plan has to be redesigned. That is why we changed the problem and model in a dynamic version, where only short term actions are taken constantly depending on the newest available information. That includes the use of the failure time distribution as much as possible/available to directly react on new situations and adapt if necessary. Moreover, we assume the use of complex assets that can submit alerts when realising that a failure is bound to happen. These alerts are also used to have a better estimate again of when a failure can happen and for that reason to rethink the policy of the traveling engineer dynamically. The goal of our research is: *Find an effective policy to the Dynamic Traveling Maintainer Problem with Alerts*.

Given the model, we introduce three different approaches. A first approach tries to make a greedy ranking of all assets that require maintenance, based on either distance, or expected failure time or expected cost. A second approach returns to the static solution approach but inputs the uncertainty as an extra variable, which should improve the policy. Finally, we introduce a sophisticated reinforcement learning algorithm that should be able to learn how the uncertainty works and how the engineer's policy should be adapted to this.

We introduce a case study for which we show how well the different approaches work compared to the optimal solution. The author of this thesis was responsible for the greedy heuristics and modeling the problem to find the optimal solution.

1.5. Thesis Outline

This thesis presents four chapters that focus directly on the optimization of DIPs and a chapter that solves the more specific Dynamic Traveling Maintainer Problem. The chapters 2 and 3 introduce a solution for the transition from DIPs to optimization problems, where in Chapter 2 we introduce a basic approach on how to filter out an optimization problem from a DIP and in Chapter 3 we extend this approach to be able to optimize larger DIPs. Chapters 4 focuses on the methods and heuristics that can solve the basic optimization problems that are introduced in Chapter 2. Chapter 5 introduces an extension both for the problem and the solution. In Figure 1.2 we have outlined the thesis chapters according to the suggested order of reading. We suggest to first read Chapter 4 as it is the most general chapter and therefore a good introduction to the other chapters. Chapter 6 can be read individually and uses a different model and solution methods for optimization.

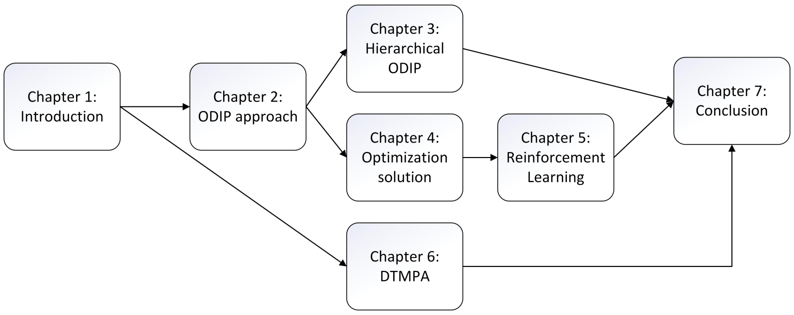

Chapter 1: Introduction
Chapter 2: ODIP approach
Chapter 3: Hierarchical ODIP
Chapter 4: Optimization solution
Chapter 5: Reinforcement Learning
Chapter 6: DTMPA
Chapter 7: Conclusion

Figure 1.2: Thesis outline

2 Optimizing Information Gathering in Decision-Intensive Processes

2.1. Introduction

Nowadays many business processes rely on knowledge workers that gather data from different sources to make informed decisions. Examples are handling complex insurance claims, developing new medicines, or diagnosing failures of high-tech machines. Such business processes require substantial flexibility [Vaculín et al., 2011], as new information may lead to new insights, hence new decisions.

A simple example of a DIP is the process of diagnosing a failure of a high-tech machine. To determine which part of the machine is failing, an engineer can collect information on software failures, do remote checks, do test runs, etc. Each piece of gathered information improves the knowledge position of the engineer and therefore decreases the uncertainty of diagnosing a wrong cause of failure. However, the time that a machine is not running is costly to the organization. Hence, an engineer needs to balance, while diagnosing, the cost and gains of gathering additional information in determining the cause of failure.

A decision maker can find themselves entangled in a web of possible actions and decisions and feel the pressure to make cost-effective decisions of good quality.

This chapter is based on Voorberg et al. [2021].

In environments where the amount of information is overwhelming, this causes problems to oversee the situation [Bossaerts and Murawski, 2017]. Still, intelligence of human decision makers, i.e., tacit knowledge, is indispensable in many situations. Hence, there is a need to improve the support for data-driven human decision making [Pomerol, 1997, Horita et al., 2017].

Next to the work on DIPs [Bromberg, 2007, Vaculín et al., 2011, Eshuis, 2018], there is a stream of related work on knowledge-intensive processes [Ciccio et al., 2015, Venero et al., 2019, Eshuis et al., 2021], in which decision-making plays a less prominent role, or decision-aware processes [Yousfi et al., 2015, 2019, Mertens et al., 2017], which do not focus on decision making by knowledge workers. Some of these works [Venero et al., 2019, 2020] stress the need for incurring data that appears during the process. However, the use of only data is not enough to optimize the actions that are taken in a DIP. The process model can raise additional constraints on actions that have to be incurred in the optimization. In the described streams of literature, this combination of information and process optimization has been recognized [Eshuis, 2018, Yousfi et al., 2019, Hasić et al., 2018]. However, no paper has given a clear approach on how to combine the information and the process model into one optimization problem.

The main modeling language for decision making in business processes is the DMN notation [Object Management Group, 2020], which declares decision rules organized in decision tables. However, decision rules specify patterns based on information without continuously optimizing the expected benefit of decisions or checking process constraints that affect those decision rules. Hence, there is the need for an approach where human decision makers can control a DIP while receiving recommendations for decisions that optimize for both the process model and the information model.

In this chapter, we define an approach that guides decision makers in making decisions in DIPs in the most efficient and effective way. The approach optimizes the costs of acquiring additional information versus the benefit of a final decision with improved quality and thus more profit. There are two sequential phases: a design-time phase and a deployment phase (Figure 2.1). In the design-time phase the decision maker is assisted in finding an optimal information-gathering solution from process models and probabilistic data. A user-defined DIP model (specified in CMMN [Object Management Group, 2015]), consisting of information

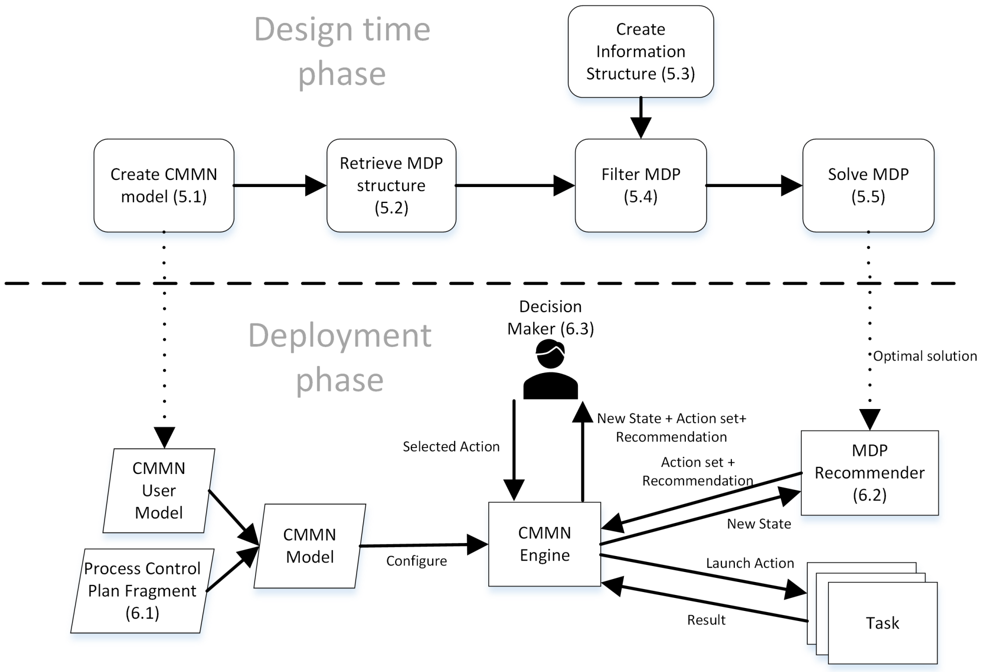


Figure 2.1: Overview of the approach; each number refers to a section

gathering tasks and decisions, is translated into a Markov Decision Process (MDP [Puterman, 2014]) with constraints. Furthermore, we introduce an information structure: a function that computes the reward for each final decision given the available information. To maximize the expected profit according to the information structure, for every state of the DIP the optimal action (gathering new information or taking a final decision) is computed by the MDP. In the deployment phase, the CMMN user model and a general process control plan fragment combine into a CMMN model that can be deployed with a CMMN engine; the deployed instance is linked with the MDP information-gathering solution in order to provide in every state a recommendation to the decision maker who controls the DIP.

As host notation we use CMMN Object Management Group [2015], since it is an OMG standard that is well suited for expressing DIPs [Marin et al., 2015]. We extend CMMN with flexible decision support in a way that goes beyond the current state of the art. Using an optimization method that takes into account both process and information we surpass the sole declaration of decision rules in DMN. The flexibility occurs when the decision maker ignores the support and takes a different action. Other than DMN, the MDP has an optimal solution also for any new situation that appears, and the approach supports such ignoring actions. The approach applies to any process modeling language in which decision makers have to decide per case which information-gathering tasks to perform.

In this chapter we make three main contributions. These contributions will be further grounded in current literature in Section 2.8. First, we define an approach that optimizes a given DIP. Second, the approach considers the data and process structure when optimizing a DIP. Third, we introduce a flexible recommendation tool that guides a human decision maker, while allowing recommendations to be ignored.

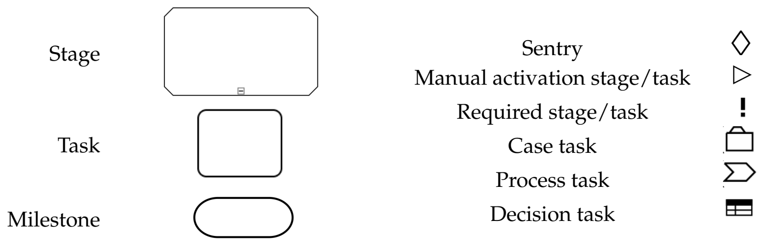
This chapter is organized as follows. In Section 2.2, we introduce CMMN as host notation for DIPs and MDP as optimization method. Subsequently, we introduce a motivating example and its CMMN case in Section 2.3. In Section 2.4, we discuss important assumptions and constraints in our approach that allow to use it in a consistent way. Section 2.5 introduces the design time steps of the approach and Section 2.6 introduces the deployment phase of the approach. Section 2.7 shows the use of the approach on the motivating example. Finally, in Section 2.8 we will discuss related work to the approach and in Section 2.9, we discuss the main assumptions that are required for the approach, respectively. The chapter is concluded in Section 2.10.

2.2. Preliminaries

2.2.1 Case Modeling Management Notation

The OMG standard Case Management Model and Notation (CMMN) Object Management Group [2015] offers a representation for semi-structured business processes in which cases are handled in a flexible way. CMMN mainly focuses on defining, in a declarative way, the flow of a case in a case plan model. The structure of the accompanying case file is assumed to be modeled by a different language, for example using a UML class diagram or an ER model. A CMMN case plan model, consists of *stages*, in which composite work is performed, *tasks*, in which atomic pieces of work are performed, and *milestones*, which are intermediate or final

Table 2.1: CMMN constructs



business objectives. These constructs are related via business rules, called sentries. Changes in the case file cause sentries to be triggered and allow tasks or actions to get enabled and started. Manual activation stages/tasks can be performed or skipped if they are enabled. CMMN can be used in combination with BPMN, for modeling structured routine business processes, and DMN, for modeling decisions embedded in processes. Table 2.1 shows the graphical notation for CMMN.

In the case of a DIP, the decision maker is constantly balancing between making progress by collecting more information or making a final decision based on the collected information. As we will show in this chapter, CMMN is well suited to express this. Especially, CMMN allows stages and tasks to be manually started or skipped (triangle symbol). However, CMMN does not provide guidance to decision makers when to perform these actions. Therefore, we introduce MDPs to consider the future effect of decisions on both the case file and the case plan.

2.2.2 Markov Decision Process

The approach that we define relies on Markov Decision Processes (MDPs) as mathematical optimization model Puterman [2014]. A Markov Decision Process is a technique that optimizes sequential decision making under uncertainty. MDPs optimize the expected reward over a series of future actions. Thus, MDPs can be used to foresee what the effects will be of taking actions now and in the future. Using backward induction (sometimes also called dynamic programming) as solution method for finite horizon problems[Puterman, 2014, p. 92], one can compute the optimal action for every state the process can be in. We specifically discuss this method for our approach in Section 2.5.5. Every *action* taken by the agent, for instance a decision maker, brings the process in a new *state* and returns a certain *reward*. The subsequent state after taking an action is defined by the transition and more specifically by the *transition probability*. Hence, the next state is uncertain and depends on a probability distribution. At the end of Section 2.3, we give an example of an MDP based on the introduced example. Section 2.5.4 gives a detailed description of how we translate a CMMN plan into an MDP model.

2.3. Motivating example

This section introduces a real-world case scenario that describes the need for decision support in information gathering. We use this example case in Section 2.7 to show the feasibility of our approach and its benefits.

Fokker Services is a company that performs maintenance on aircraft components. Airliners close service contracts with Fokker Services for a specific component over multiple years. The market of aircraft component maintenance is competitive and requires sharp offers from Fokker to bring in contracts. To know the profit margins that are available for a specific component, information gathering is crucial. Fokker engineers need support on which information is most important and still valuable enough to collect such that they win attractive contracts and avoid unattractive contracts.

The process of these service contracts works as follows. Fokker Services receives a Request for Quotation (RFQ) from a potential client, who may also send this RFQ to competitors of Fokker. The RFQ refers to an airplane component, for which the client requests a proposed contract that explains Fokker's terms for the service contract, in this scenario a specific repair price for the serviced component. After performing an internal DIP in which extra information can be gathered, Fokker sends the terms of their contract. Fokker Services either earns the contract or the client chooses the offer from a competitor. Fokker is looking for an approach to support the decision makers by informing them about the value of gathering more information and the value of current options for the conditions of the contract.

We focus on the internal DIP. At the moment that the RFQ arrives, Fokker has no exact information concerning the properties of the component that should be quoted. Fokker Services can gather information on this component and the terms and conditions of this RFQ through the data attributes listed in Table 2.2. Note that Parts

Table 2.2: Information acquiring tasks and attributes

Task	Data Attribute	Reference
Obtain net repair costs	Net repair costs	A
Obtain Expected number of repairs	Expected number of repairs per year	B
Check customer credibility	Customer credibility	C
Check market prices	Current mean market price	D
In-house capability check	In-house capability	E
Determine PMA/DER	PMA/DER possibilities	F
Determine over and aboves	Over and above options	G
Fix contract length	Contract period	H
Obtain transport costs	Transportation costs	I

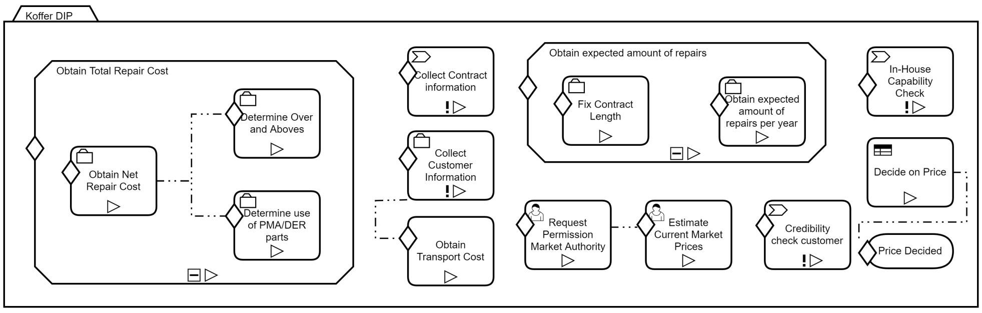

Koffer DIP
Obtain Total Repair Cost
Determine Over and Aboves
Obtain Net Repair Cost
Determine use of PMA/DER parts
Collect Contract information
Collect Customer Information
Obtain Transport Cost
Obtain expected amount of repairs
Fix Contract Length
Obtain expected amount of repairs per year
Request Permission Market Authority
Estimate Current Market Prices
Credibility check customer
In-House Capability Check
Decide on Price
Price Decided

Figure 2.2: CMMN plan for Fokker Services

Manufacturer Approval (PMA) and Designated Engineering Representative (DER) are methods that suppress the costs of a component by using cheaper alternatives. Over and above parts are excluded from the contract and can be charged separately.

Fokker Services aims to optimize the process of gathering information through these attributes and hence to increase their certainty about how much profit they can make on this component. This increased certainty improves the effectiveness of their quote price. At the same time, Fokker loses efficiency because of valuable time that is lost while gathering information. Other, possibly more profitable, RFQs might not be handled as long as this RFQ is still the main focus. Hence, they need a DIP to optimally make the trade-off between time and information. Figure 2.2 shows the CMMN plan for the internal DIP. This figure contains more tasks than the information-acquiring tasks of Table 2.2, since the entire DIP includes other tasks as well.

Figure 2.3 shows the Markov Decision Process that is the translation of the Obtain

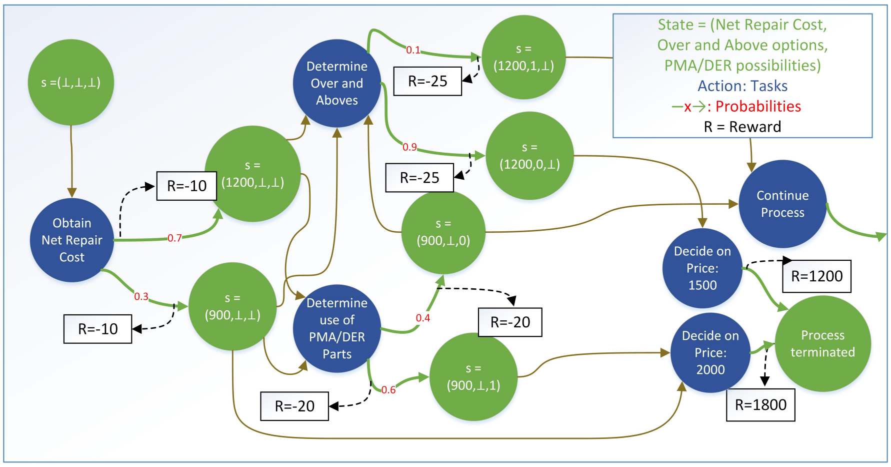


Figure 2.3: Markov Decision Process

Total Repair Costs substage from the CMMN plan in Figure 2.2, according to the translation defined in Section 2.5.4. The *state* defines which tasks have been performed and what attribute information has been collected, where ⊥ means the information is unknown. In Figure 2.3 we only show the information. One takes an *action* by deciding which subsequent task to perform. Every action causes a change of process (case file) or information (case plan) status. *Transition probabilities* determine the unknown outcome of information acquiring tasks. By putting cost (negative *reward*) on performing more tasks and returning a reward based on how good the final decision is, we can use the optimized policy to find the best action such that the expected profit is maximized.

2.4. Optimizable DIPs

In this chapter we focus on DIPs with a specific structure that allows linking them to Markov Decision Processes (MDPs) for optimization. Hence, we call them *optimizable DIPs*. In Section 2.4.1, we set out what is the specific constraint for optimizable DIPs. In Section 2.4.2 we discuss how CMMN is used to model these *optimizable DIPs*.

2.4.1 Stable information

In a DIP information is gathered to reach the end goal in an optimal way. In this chapter we use data attributes to model the information needs of a DIP. In the class of optimizable DIPs, we assume acquired information to be accurate and *stable*, i.e., values acquired for data attributes are assumed to be correct and are not changed once acquired. The assumption that data attributes are stable allows us to use them to decrease uncertainty on the received reward. Attribute values are retrieved by performing tasks. If a DIP only has stable data attributes, tasks are monotonic: if a task writes an attribute it is performed at most once during execution.

Before executing the DIP, the value of a data attribute X can be estimated using a probability distribution, $P(X = x) \in [0, 1]$, where x is a value for X. The probability distribution could be derived for example by analyzing historical data or by interviewing domain experts. For each execution of a DIP with n data attributes, the set with data attributes is denoted by $D = \{X_1, \ldots, X_n\}$. For each data attribute $X_i \in D$ we assume a domain $dom(X_i)$ of possible values for X_i. The space of data attribute values is denoted by $\mathcal{D} = dom(X_1) \times \ldots \times dom(X_n)$. Using a probability distribution for each data attribute allows us to know the set of possible values and also the relative appearance of every possible value. The probability distributions enable to estimate the different outcomes (variability) of a specific decision. Also, one can compare different future scenarios where more information becomes available, which would improve the certainty of decisions by decreasing the variability of possible outcomes. Hence, knowing these distributions enables to determine if data attributes can provide extra information for a final decision.

2.4.2 Plan Constraints in CMMN

Although the main trade-off for an optimizable DIP lies in the data attribute value gathering (information perspective), we are considering this problem from the perspective of a business process. As such, many different constraints might prevent a decision maker from taking the most direct path in collecting certain attribute values. For example, there might be subtasks that must be performed first before one can continue with other tasks. These constraints are modeled in the CMMN process model, for instance required tasks, precedence constraints between tasks, and stages that contain tasks. Furthermore, we allow for tasks in the CMMN model that are part of the procedure but actually do not affect the decision making part of the DIP. Our approach, in particular the MDP, deals with all these additional constraints. This means that we do not only optimize from an information-perspective (case file), but also from a process-perspective (case plan).

While we allow for plan constraints, we also make some simplifying assumptions in this chapter. These assumptions are necessary to filter out a tractable MDP model (cf. Section 2.5.4). First of all, our algorithm does not support milestones. Therefore, any milestones that are currently modeled in a CMMN model will be excluded, except for the milestone that models the achievement of the main goal of the whole case. In Section 2.9 we discuss how this assumption can be relaxed. Furthermore, we assume that tasks are instantaneous, so upon invocation they return immediately an outcome (data attribute value). Non-instantaneous tasks lead to an explosion of the state space of an MDP.

2.5. Dynamic decision support for optimizable DIPs: Design time

In this section, we define the design-time steps of our approach to optimize the process of information gathering in Decision Intensive Processes. The upper part of Figure 2.1 shows a flowchart of this part of the approach, consisting of five steps. These steps translate the DIP into an optimizable MDP model and solve it. Section 2.6 explains how we use the solved MDP to configure a recommender that offers decision support during deployment.

2.5.1 Create CMMN model

The first step is modeling a DIP with CMMN notation. A CMMN process model is called a case plan model, which consists of a top-level stage that contains stages, tasks and milestones. CMMN calls these elements plan items, and distinguishes between the definition and the use of plan items. However, we assume that each definition is used only once in a case plan model, hence we equate plan item definitions and plan items. Note that this assumption follows directly from the stable information assumption in Section 2.4.1, causing tasks to be monotone. We allow the use of all possible CMMN task types including human tasks, case tasks and process tasks. For example, in Figure 2.2, tasks such as Define Over and Aboves or Collect Customer Information are case tasks, while Estimate Current Market Prices is a human task. However, since only the output data of a task is needed to make decisions, the specific CMMN task type can be ignored. CMMN is suited for dynamic data where stages or tasks can be visited multiple times to update data items. However, because of the stable information assumption (Section 2.4), the stages and tasks in our DIP are not reactivated after their completion. Tasks for which the decorator (!) is used are *required*: these model elements need to be finished before the parent stage can be completed.

This chapter focuses especially on tasks and stages that need to be activated manually by the user, i.e., decision maker. The approach helps the user to decide, while enacting the CMMN process model, whether and when to activate such a task or stage. Note that a manually activated task or stage can have an entry criterion, that needs to evaluate to true before the manual activation decision can be made. The approach allows that for some task or stage automatic activation is used. However, to keep the interpretation of the model clear, we have not implemented this in our example case. We only consider entry criteria having sentries with **on** statements. To keep the model representation interpretable, sentries having **if** statements are not considered. However, MDPs do allow for the use of conditions on the data through **if** statements.

Data in a CMMN case is stored in a case file, which contains case file items. Therefore, the *case file* will denote the set of data attributes that can be acquired. To simplify the presentation, we allow a task to be connected to at most one case file item. However, this constraint can be relaxed by taking the joint probability distribution of the case file items for that specific task. To see an example of a CMMN model that is according to the optimizable DIP description, we refer to Figure 2.2.

2.5.2 Retrieve MDP Structure

To be able to continue our approach, we require that all tasks are assigned to one of four classes: three classes of tasks and a class of stages. Furthermore, we require the presence of a single milestone that terminates the process. Figure 2.4 is a meta

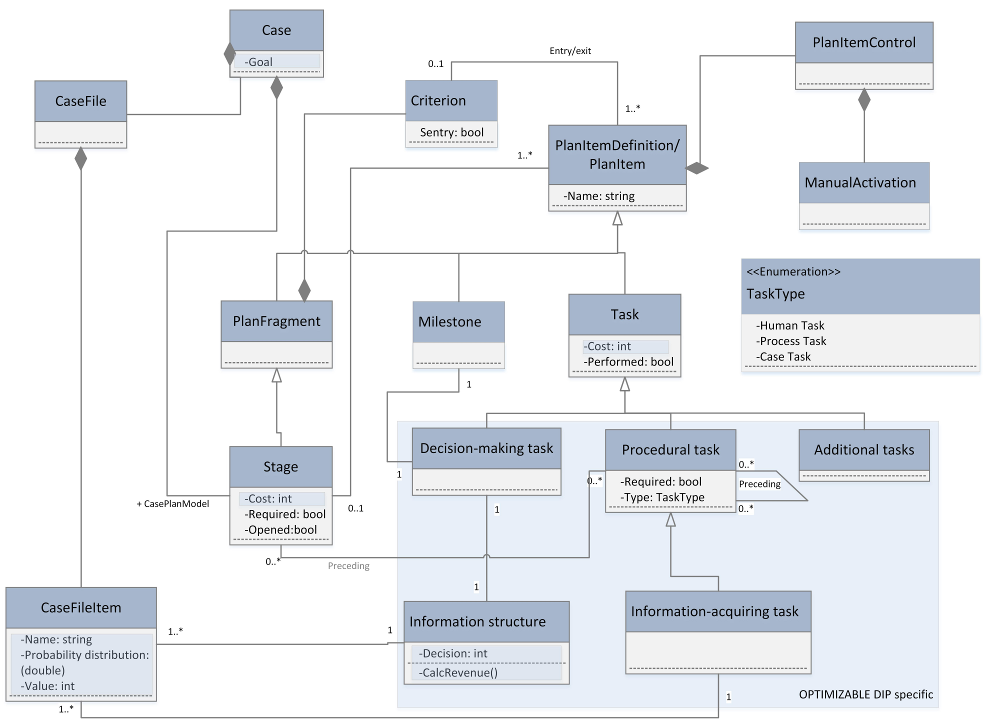


Figure 2.4: CMMN meta model Grudzińska-Kuna [2013], Object Management Group [2015] extended with support for DIPs (in light blue)

model in UML notation that shows how these different classes are associated with each other and relate to the CMMN classes. The class diagram also contains the class of case file items with their known probability distribution.

Tasks Consider the complete set of tasks T also introduced in Figure 2.4. For every task $t \in T$ we assign an associated costs c_t, representing the costs and effort of performing the task, and a boolean variable that keeps track of whether the task has been performed.

Each DIP works towards a *final decision*. We model this final decision with a decision making task. We assume a DIP has one such decision making task, that is not followed by other tasks. However, in Section 2.9, we come back to this assumption and give suggestions on how the approach could be extended with multiple (sub)decisions in one DIP.

Definition 1 (Decision making tasks/Milestones). *A decision making task uses acquired information from the case file to choose a decision f from a set of decision alternatives $\mathcal{F}$. The decision finishes the DIP by achieving its final milestone.*

We define the decision making task $d \in T$. In accordance with the UML class diagram in Fig. 2.4, a decision making task d is associated with an information structure (Section 2.5.3) and a single milestone and inherits two attributes from the task class. In the CMMN models, we interpret decision (DMN) tasks to be decision making tasks. Moreover, a decision making task is always followed by a milestone. Normally an underlying DMN model would define the decision tree for a decision task, but in our approach we define an MDP that governs the decision through a look up table that is introduced in Section 2.6.

Definition 2 (Procedural tasks). *A procedural task is a task that is performed to progress towards the final decision of the DIP. It has a direct effect by being either required(1) or preceding(2) or part of a stage(3) or information acquiring(4).*

We define the set of procedural tasks $P \subset T$. In the example we see the task of Requesting Permission Market Authority preceeds task Estimate Current Market Price. Hence, the permission is a preceding task(2) in this example, which makes it a procedural task. We define procedural tasks that collect one of the attribute values of unknown attributes as information acquiring tasks, $I \subseteq P$.

Definition 3 (Information Acquiring tasks). *An information acquiring task is a procedural task that updates one or more attributes in the case file when it finishes.*

An example of an information acquiring task is the In-House Capability check in Figure 2.2. Every information acquiring task has a known association with one case file item and these tasks also inherit the process-related class attributes for procedural tasks.

Besides decision making and procedural tasks, a DIP can have additional tasks. Although they may have a certain costs associated, they do not directly affect the decision in the process and thus are negligible for the MDP and the approach. Therefore, any additional tasks are discarded. Discarding those tasks for the decision-making model does not mean those tasks are removed from the business process.

Definition 4 (Additional tasks). *An additional task is a task that is in the CMMN model and is neither a decision making nor a procedural task.*

Stages Stages $v \in V$ create the hierarchy inside a plan model. We use stages to group certain clusters of tasks. The stage is an identifier of a cluster of tasks. Similar to a task if a stage is in the *preceding* set of another stage or task, it prevents the task/stage inside the cluster from starting if it has not been opened yet. Due to their structure, for the MDP model we can interpret stages as a special class of procedural tasks.

Labeling tasks Based on the UML class diagram and using the constraints for the optimizable DIP, we can recognize and retrieve all the above defined classes of tasks and stages automatically. The decision making task is attached to the only milestone in the plan. Furthermore, all tasks that do not fulfill any constraints to other tasks and are not associated with an attribute are discarded as additional tasks. The remaining tasks are procedural. Any of these procedural tasks that have an association with an attribute are classified as an information acquiring task. Similar to procedural and additional tasks, stages can be filtered naturally from the starting CMMN model and do not require additional verification by the plan builder. Note that the Fokker case according to Figure 2.2 does not contain any additional tasks.

The introduced task types resemble activity types that have been proposed for making decisions in processes Hasić et al. [2018]: Decision making tasks resemble decision activities, information acquiring tasks resemble administrative activities and additional tasks resemble operational activities. However, procedural tasks with procedural constraints (required, precedence) have no counterpart in Hasić et al. [2018].

2.5.3 Information Structure

A DIP is performed to achieve a specific functional goal that relates to a decision to obtain the best outcome. This *final decision* is the crucial step that affects the outcome of the whole process. For example, in the Fokker Services case in Section 2.3, the decision on the quoted price is the *final decision*. While each DIP has a specific functional goal, i.e., an outcome, a final decision should be both efficient and effective, which are non-functional goals. Since both non-functional goals are in conflict, this requires a trade-off between them. Note that the literature on DIPs or related Knowledge-Intensive Processes (Section 2.8) considers only functional goals and not the trade-off between non-functional and functional goals to reach an optimal decision.

To optimize for decision efficiency, one could set constraints on, for example, the total effort spent on the process. However, these predefined constraints could affect the search for optimality in a negative way. For example, if a constraint permits to only collect a single piece of information, one cannot react in a flexible way to the situation. Hence, it is preferable to be able to compare efficiency and effectiveness in a direct way. Therefore, we propose to define a cost factor for the time spent on different tasks. This way, we can also distinguish better between tasks that are cheap (simple task) versus expensive (difficult task). Together with a reward function based on the process outcome, it is then possible to make this direct comparison. We next introduce mathematical notation for specifying how to optimize the process. We require a function that connects the effect of acquired data on final decisions and the costs of collecting that data to a reward. More precisely, the input of this function includes a final decision f from a set of alternatives $\mathcal{F}$, which must be chosen to terminate the process, and all the data attribute values $\mathcal{D}$, with a known value. We call this function the information structure. In earlier work Voorberg et al. [2019], we already introduced a first definition for this information structure. We extend that definition here, such that it also optimizes over the decision variable that directly determines the outcome of the process.

Definition 5 (Information Structure)**.** *Given a set of data attributes $D = \{X_1, \ldots, X_n\}$ with possible values $\mathcal{D} = dom(X_1) \times \ldots \times dom(X_n)$ and a set of final decision alternatives $\mathcal{F}$, the* ***information structure*** *is a function $Y : \mathcal{D} \times \mathcal{F} \to \mathbb{R}$, which gives the reward given all information and the chosen alternative. We denote by $Y(X_1, \ldots, X_n, f)$ a function that gives the reward, where the data attributes are random variables.*

The method CalcRevenue() in Figure 2.4 is the actual information structure Y that based on the attribute values and the set of decision alternatives calculates the optimal decision for the decision making task. An information structure can be derived through a combination of data analysis and domain knowledge. If there is a set of historical cases of this process, where preferably all data attributes, the decision and the outcomes are known, one could use techniques such as regression to estimate the effect of parameters on the outcome, while taking into account the decision options, as we discuss in Voorberg et al. [2019].

We give a small example to show what the information acquisition can do with the outcome of the process. Suppose we have two data attributes, X_1, X_2, for which the values are unknown: $P(X_1 = 0) = P(X_1 = 3) = 0.5$, $P(X_2 = 1) = P(X_2 = 4) = 0.5$. The two decision alternatives that can be taken are 0 (decline) or 1 (accept): $\mathcal{F} = \{0, 1\}$. The information structure is attribute 2 minus attribute 1 multiplied by the final decision: $Y(X_1, X_2, f) = (X_2 - X_1)f$. The acquisition costs are 0.5 per attribute: $c_1, c_2 = 0.5$. Suppose the decision maker first wants to explore the acquisition of attribute 2. If $X_2 = 4$, we know the optimal final decision, independent of the value of X_1: $f = 1$. Then, knowing that $P(X_1 = 0) = P(X_1 = 3) = 0.5$, there is an expected profit of $\mathbf{E}[Y] = (0.5 \cdot (4 - 3) + 0.5 \cdot (4 - 0)) \cdot 1 = 2.5$. However, when $X_2 = 1$, the optimal decision is: $f = 0$, because $\mathbf{E}[Y] = (0.5 \cdot (1 - 3) + 0.5 \cdot (1 - 0)) \cdot f = -0.5 \cdot f$. Hence, the optimal profit $\mathbf{E}[Y] = 0$ for $f = 0$. Similarly, we can derive that without any information the optimal decision is $f = 1$, with an expected profit of 0.5. To interpret the value of attribute 2, we compare the profits when knowing and not knowing the value of X_2. For attribute 2: we have $(0.5 \cdot 2.5 + 0.5 \cdot 0) - 0.5 = 0.75$ as profit increase, which exceeds the costs c_2. So, it does pay off to acquire the information for attribute 2 right now.

In conclusion, executing a DIP in the most efficient and effective way can be viewed as a problem of profit maximization. Collecting less information can decrease the cost of the process. However, the lack of information creates uncertainty about the outcome of the final decision. By collecting more information this uncertainty can be decreased. The information structure is the solution to consider both factors by making a trade-off between extra costs versus improved revenue.

2.5.4 Filter MDP model

Given the information structure and the assignment of tasks to the specific classes, one can now filter out an MDP model. In this chapter, we focus on the approach and not on the specific mathematical details of filtering out an MDP.

For the CMMN model, the MDP state $\mathbf{s}$ of the DIP is a vector that contains for each variable its value. A variable is either a case file item (data attribute) or a status attribute of a task or stage (case plan item). During the execution of the DIP each stable data attribute X_i, $1 \leq i \leq n$ begins with an unknown value, expressed with symbol $\perp$. The actual value x_i for X_i can be retrieved. Together the data attribute variable i has a state $s_i \in dom(X_i) \cup \{\perp\}$. A vector with data attribute values, both known and unknown, is interpreted as an information state, denoted with $s = (s_1, \ldots, s_n)$. Furthermore, the total space of possible states is denoted by $\mathcal{S} = (dom(X_1) \cup \{\perp\}) \times \ldots \times (dom(X_n) \cup \{\perp\})$. For each procedural task or stage $y \in P \cup V$, we use a binary status attribute σ_y that has value 1 if and only if the task or stage y has finished: $\sigma = (\sigma_a, \sigma_b, \ldots, \sigma_{|P \cup V|})$. We call σ the plan state. The total state of the process is $\mathbf{s} = (\sigma, s)$.

For the information acquiring tasks to have the performed variable set to true, we require the associated case file items to have a known value. Based on this, the decision maker can choose from two different sets of actions. First, the decision maker can choose to perform a procedural task and possibly acquire an uncertain attribute, i.e., $\mathbf{a} = p \in P$, which changes the state because the task performed attribute is set to true. Notice that although we do not include the exact value of the acquired attributes information in the plan state, this value does have a crucial effect on the outcome of the process as part of the information state s. Second, the decision maker can decide to make a final decision, $\mathbf{a} = f \in \mathcal{F}$. In that case we end up in a terminating state because the DIP is finished. Notice that the opening of stages and required or preceding tasks puts some extra constraints on the actions allowed.

Let $preceding_{\mathbf{a}}$ be the set of procedural tasks or stages that must have been performed before task or stage $\mathbf{a}$ is allowed. In Figure 2.2 this is denoted by a dotted line that connects the two tasks. Finally, let $required_{\mathbf{b}}$ be the boolean variable that denotes if task $\mathbf{b}$ must be performed before the process can be finished. Then, for the decision maker to take action $\mathbf{a}$, we require the following conditions to hold:

$$\forall \mathbf{b} \in preceding_{\mathbf{a}} : \sigma_{\mathbf{b}} = 1 \qquad \text{(Preceding tasks or stages)}$$

$$\text{if } \mathbf{a} \in \mathcal{F} : \forall \mathbf{b} \in P \cup V, \text{if } required_{\mathbf{b}} = 1, \text{then } \sigma_{\mathbf{b}} = 1 \qquad \text{(Required tasks or stages)}$$

As long as the decision maker takes information acquiring actions, we incur costs for the specific task that is performed. The moment the decision maker decides to make a final decision, we retrieve an expected reward based on the currently known attribute values and the decision taken. Hence,

$$R(s,\mathbf{a}) = \begin{cases} -c_{\mathbf{a}} & \text{if } \mathbf{a} \in P \\ \mathbf{E}[Y(s,\mathbf{a})] & \text{if } \mathbf{a} \in \mathcal{F} \end{cases}$$

Note that we can take the expectation over Y as it has been defined also for the set of random variables D. For a state s where some information is already revealed, the expected reward of taking final action a can be expressed as a conditional expectation as follows $\mathbf{E}[Y(s,\mathbf{a})] = \mathbf{E}[Y(X_1,\ldots,X_n,\mathbf{a})|\forall i : x_i = s_i \text{ if } s_i \neq \perp]$. This takes into account the known attribute values, while using the distribution for the remaining attributes.

Transitions happen with a certain probability that can be derived from the probability distributions that we establish when classifying the information-gathering tasks. We define two transitions: state transitions (case plan) and information transitions (case file). The total transition of an action **a** in the DIP constitutes of the product of both transitions. Looking back at the defined classes of tasks, only in information acquiring tasks we have an information transition. In the other classes the information transition probability will always be 1.

For state transitions we have the following:

$$P^p_{\mathbf{a}}(\sigma,\sigma') = \begin{cases} 1 & \begin{array}{l}\text{if } \mathbf{a} \in P \cup V \text{ and } \sigma_{\mathbf{a}} = 0 \text{ and } \sigma'_{\mathbf{a}} = 1 \\ \text{and } \forall \mathbf{p} \in P \setminus \{\mathbf{a}\} : \sigma_{\mathbf{p}} = \sigma'_{\mathbf{p}} \qquad \text{(Procedural task)}\end{array} \\ 1 & \text{if } \mathbf{a} \in \mathcal{F} \text{ and } \forall \mathbf{b} \in P \cup V : \sigma_{\mathbf{b}} = \sigma'_{\mathbf{b}} \qquad \text{(Decision-making task)} \\ 0 & \text{otherwise} \end{cases}$$

For information transitions we define the following, where $\leftrightarrow$ means an information-acquiring task and case file item are associated:

$$P^c_{\mathbf{a}}(s,s') = \begin{cases} P(X_i = x_i) & \text{if } \mathbf{a} \in I: \text{ if } i \leftrightarrow \mathbf{a} : s_i = \perp \text{ and } s'_i = x_i \text{ and for } j \not\leftrightarrow \mathbf{a} : s_j = s'_j \\ 1 & \text{if } \mathbf{a} \in \mathcal{F} \cup V \cup P \setminus I \text{ and } 1 \leq i \leq n : s_i = s'_i \\ 0 & \text{otherwise} \end{cases}$$

Hence, the total transition probability is then:

$$P_{\mathbf{a}}(\mathbf{s},\mathbf{s}') = P^p_{\mathbf{a}}(\sigma,\sigma') \cdot P^c_{\mathbf{a}}(s,s')$$

By collecting extra information, it is possible to reduce the uncertainty on the final decision outcome. Hence, there is a higher certainty that one actually will earn what is expected. However, to arrive at that point, efficiency is lost while acquiring additional information. The trade-off between collecting more information and maximizing the expected reward based on the final decision is the key parameter that this MDP tries to maximize by scaling those rewards to their probability of happening. One needs to know the probability distributions of the individual attributes because they allow to compare the change in profit between the attributes based on their probabilities.

2.5.5 Solve MDP

To solve a finite horizon MDP, we can use the technique of backward induction. However, this is only the case in scenarios where the state space still is tractable in computer memory. Backward induction starts from all the possible scenarios that the process could end up in and their reward. By thinking backwards what would have been the best action to maximize reward if still one variable would be uncertain, we set one step back in time. This becomes a recursive procedure all the way back until all attributes are uncertain. During this backward induction, every possible state is visited. This also means that for every possible state, we have determined what is the optimal action to take. To know in detail how backward induction works for a MDP based on an optimizable DIP, we refer to our earlier work Voorberg et al. [2019], which builds on existing MDP solution techniques Puterman [2014]. Note that the state space of an MDP grows exponentially. This means that we can solve problems up to the size of the example case. Hence, scalability is an important future extension, which we discuss in the conclusion. An MDP always optimizes for the expected profit. However, by retrieving information one also decreases the uncertainty of the outcome. Because in this chapter we dot not put any conditions on the information structure, this function can become very complicated with many dependencies between attribute values. Therefore, we cannot give any further indication on the variance reduction for different information-acquiring decisions.

2.6. Dynamic decision support for optimizable DIPs: Deployment

The outcome of our approach for the design-time phase is a solved MDP that offers an advice for every possible state in which the DIP can be. We now define how to use these MDP advices during deployment. The lower part of Figure 2.1 shows how a decision maker controls a CMMN engine while interacting with the MDP recommender. For this purpose, we introduce a plan fragment to control the execution of the user-defined CMMN process model. A plan fragment is a reusable part of a case plan model and often contains multiple plan items. Inserting the plan fragment into the CMMN user model results in an integrated CMMN model. The CMMN engine is configured with this integrated CMMN model to let the decision maker control the DIP and launch tasks. After every task completion, the result is incurred in the new state that is sent to the MDP recommender to obtain a new action set and recommendation. Next, both the new state and the action set and recommendation are sent to the decision maker, who selects one of the actions.

2.6.1 Process Control Plan fragment

Figure 2.5 shows the process control plan fragment that fits the constraints of an optimizable DIP and its context, the CMMN user model. The figure shows the *final decision* point as defined by Decision-making task; in the example in Figure 2.2 this would be task Decide on Price. Furthermore, when deciding to gather more information the corresponding task from the CMMN user model is performed (cf. Figure 2.2).

The actual need for a plan fragment and the extra tasks is because of the effectiveness versus efficiency trade-off. This trade-off only appears after the DIP has been modeled according to our approach. Hence, when performing the DIP the decision maker needs a recurring process phase where he actually makes the trade-off supported by the MDP recommender.

The plan fragment consists of a task Request MDP recommender which results in a recommendation and subsequently the task Information or Decision where the decision maker can consult the MDP recommendation and decide on the next step.

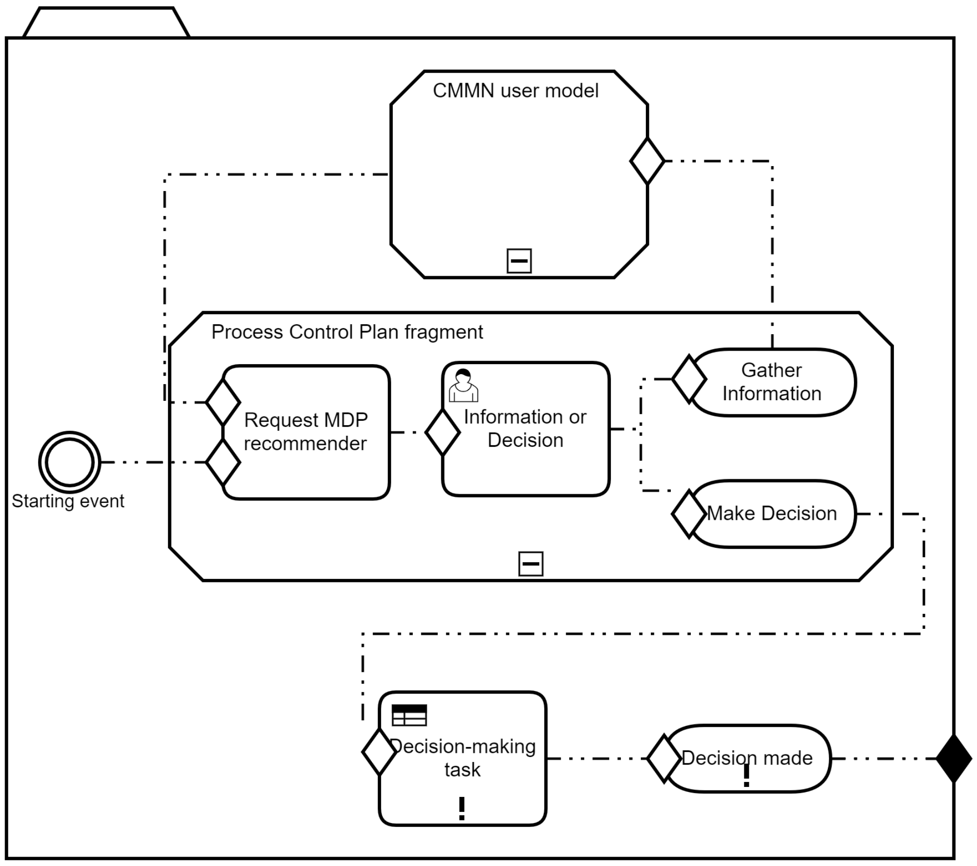


Figure 2.5: Process Control Plan Fragment for user CMMN model

The two milestones capture the decision maker's decision and control the next phase of the integrated plan. By integrating the plan fragment with the user model, an integrated CMMN model is obtained that configures a CMMN engine supporting the DIP.

2.6.2 Decision making support

The outcome of solving the MDP can be seen as a multi-dimensional matrix that functions as a look-up table for decision makers. During the Information or Decision task, the decision maker makes a decision, given the current state and MDP support, i.e., an optimal action and the corresponding expected profit for that action. Also other possible actions and their expected profit are shown to the decision maker. Section 2.7.3 will give an example of how the information could be shown to the decision maker.

The MDP support consists of three parts. Those parts are components of the recommendation handed to the decision maker in the Information or Decision task in Figure 2.5. First, the decision maker is given a list of allowed actions according to the current state of the CMMN user model. Second, the MDP advises the optimal next action, either a procedural task, e.g., which task from the CMMN user model in Figure 2.5, or a final decision. Note that for specific scenarios, we first might need to perform a 'non-acquiring' procedural task before we can actually perform the information acquiring task that gives us the required information. In that case the MDP will first advice the non-acquiring task. Third, the MDP shows the expected profit for all possible actions. This comparison was discussed in Section 2.5.3. As long as additional information still offers more value in expectation, the recommendation will be to continue information gathering.

2.6.3 Decision maker

As shown in Figure 2.1, the decision maker is the responsible process controller, i.e., the outgoing arcs of the Information or Decision task are based on a decision from a decision maker who is assisted by the MDP recommender. The inserted plan fragment then allows the CMMN engine to control the process being enacted. When the task is completed, the CMMN engine calls the recommender for a new state-specific advice, even when the previous advice was not followed by the decision maker. This new advice is then shown to the decision maker. This loop continues until the decision maker takes a final decision, after which the CMMN engine will terminate the process. However, tacit knowledge of decision makers cannot be captured in a mathematical model. Our approach therefore allows a decision maker to ignore recommendations and acquire information that is not most profitable according to the MDP. Still, the MDP offers advice in any resulting state.

2.7. Evaluation/Case study

Based on the example DIP case introduced in Section 2.3, we next show a proof of concept for our approach and validate its feasibility. Furthermore, the outcome of the MDP for the example case is compared to the current decision method within Fokker Services. Via this link: `OptimizingInformationGathering`, a second case is made publicly available, where we focus on optimizing failure diagnosing and dispatching of maintenance engineers.

2.7.1 Validation case

The Fokker case that was introduced in Section 2.3 is based on a real-world case. The validity of this model was discussed with actual decision makers for this specific kind of processes at the company. Also the data set-up, though simplified to four possible values per attribute to maintain tractability and also in simplified in terms of cost parameters, has been retrieved from this company. Finally, the decision tree that we will use as benchmark was developed in cooperation with the same decision makers.

2.7.2 Design-time approach

Create CMMN model For the first stage of our approach, we require a CMMN model of the optimizable DIP according to the constraints defined in Section 2.5. For the example DIP we already introduced the CMMN model in Figure 2.2, which does satisfy the required optimizable DIP set-up according to the constraints introduced in Section 2.4.

Retrieve MDP structure The next step that is necessary to create an MDP is to assign all necessary tasks in the CMMN model to one of three subsets. How to assign tasks to these sets follows directly from the CMMN model in Figure 2.2. Table 2.3 shows the stages and procedural tasks and their associations with other tasks including also the information acquiring tasks and the associated case file items. Table 2.4 defines the case file containing all case file items. There are no additional tasks.

	Stages	Costs	Preceding	Required	
1.	Fokker DIP	0	()	0	
α.	Obtain total repair costs	20	()	0	
β.	Obtain expected amount of repairs	0	()	1	
	Procedural tasks	Costs	Preceding	Required	
2.	Request permission market authority	20	()	0	
3.	Collect contract information	10	()	1	
4.	Collect customer information	30	()	1	
	Information acquiring tasks	Costs	Preceding	Required	Case File Item
5.	Obtain net repair costs	10	(α)	0	A
6.	Obtain yearly amount of repairs	8	(β)	0	B
7.	Check customer credibility	2	()	1	C
8.	Estimate current market prices	100	(2)	0	D
9.	In-house capability check	3	()	1	E
10.	Determine PMA/DER	20	(α, 5)	0	F
11.	Determine over and aboves	15	(α, 5)	0	G
12.	Fix contract length	4	(β)	0	H
13.	Obtain transport costs	12	(4)	0	I

Table 2.3: Procedural tasks for Fokker Services case

	Case File items	Distribution values	Distribution
A.	Net repair costs	{1500,1700,1900,2100}	Uniform
B.	Expected number of repairs per year	{40,60,80,100}	Binomial(0.2)
C.	Customer creditworthiness	{0,1}	Bernoulli(0.3)
D.	Current mean market price	{1700,1900,2100,2300}	$[\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}]$
E.	In-house capability	{0,1}	Bernoulli(0.6)
F.	PMA/DER possibilities	{100, 200}	Bernoulli(0.3)
G.	Over and above options	{0,1}	Bernoulli(0.5)
H.	Contract period	{1,2,3,4}	$[\frac{1}{6}, \frac{2}{6}, \frac{2}{6}, \frac{1}{6}]$
I.	Transportation costs	{50,100,150,200}	Uniform

Table 2.4: Case File items for Fokker Services case

To retrieve the MDP structure of the example DIP we need an information structure for the decision-making task. We define the following information structure, based on the case file items introduced in Table 2.4 and the Fokker Services case,

$$Y(s,f) = E \cdot \overbrace{B \cdot H}^{\text{Expected number of repairs}} \cdot \overbrace{P(\mathcal{D} \geq f)}^{\text{Quote accepted}} \cdot \overbrace{[f - A(1-0.2G) - F - I]}^{\text{Expected Revenue}} \quad \forall f \in \mathbb{R}$$
$$Y(s, \text{No bid}) = 0$$

The random variable $\mathcal{D}$ is modeled by a normally distributed random variable with $\mu = D$ and $\sigma = 200$. Furthermore, we have decision alternatives, $\mathcal{F} = \{\text{No bid}, 1800, 2000, 2200\}$.

Filter MDP model The next step in our approach is to build up the MDP model from the information structure and the tasks assigned to different classes. Following the approach in Section 2.5.4, we filter the following plan state which is a tuple of tasks 2 to 13 and stages α and β. In this state, the σ parameter is the 'completed' attribute for tasks and stages: $\sigma = [\sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7, \sigma_8, \sigma_9, \sigma_{10}, \sigma_{11}, \sigma_{12}, \sigma_{13}, \sigma_\alpha, \sigma_\beta]$. Note that besides this case plan state, there is also an information state that contains the values of the case file items, where x is the data attribute value: $s = [x_A, x_B, x_C, x_D, x_E, x_F, x_G, x_H, x_I]$. The action space consists of two sets. Either we collect more information by performing one of the twelve procedural tasks or we open a stage, or we take a final decision:

$$\mathcal{A} = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, \alpha, \beta\} \cup \{\text{No bid}, 1800, 2000, 2200\}$$

Table 2.3 lists precedence and required constraints for the information gathering actions to become enabled in a specific state.

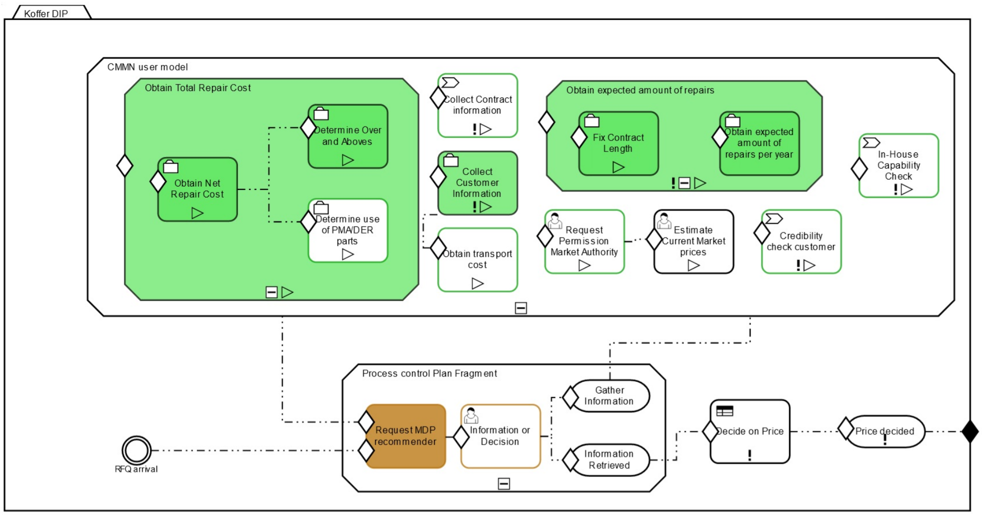
Next, the reward for a given state and action is either the extra costs for gathering information, given in the tables, or the expected profit based on the information that is available in the current state **s** and final decision $f \in \mathcal{F}$:

$$R(s, \mathbf{a}) = \begin{cases} -c_{\mathbf{a}} & \text{if } \mathbf{a} \in P \cup V \\ \mathbf{E}[Y(s, f)] & \text{if } \mathbf{a} = f \end{cases}$$

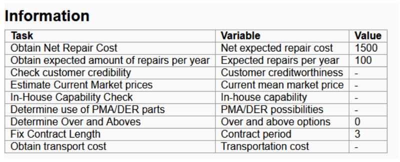
Finally, the transitions can be derived exactly the way they were introduced in Section 2.5.4 using the probability distributions in Table 2.4. As described in Section 2.5.5, the filtered MDP can be solved using backward induction techniques. We used Python to program an algorithm that optimizes MDPs that are built from tables such as Table 2.3 and 2.4.

2.7.3 Deployment

After all the design-time steps have been taken as explained above, the process to be enacted can be deployed. Having the MDP solution, the decision plan fragment is combined with the user CMMN plan and enacted by the CMMN engine to support the decision maker in performing the DIP. Figure 2.6(a) shows the current state of the case, where green filled tasks have been performed and green lined tasks are


Koffer DIP
CMMN user model
Obtain Total Repair Cost
Obtain Net Repair Cost
Determine Over and Aboves
Determine use of PMA/DER parts
Collect Contract information
Collect Customer Information
Obtain transport cost
Obtain expected amount of repairs
Fix Contract Length
Obtain expected amount of repairs per year
In-House Capability Check
Request Permission Market Authority
Estimate Current Market prices
Credibility check customer
Process control Plan Fragment
Request MDP recommender
Information or Decision
Gather Information
Information Retrieved
Decide on Price
Price decided
RFQ arrival

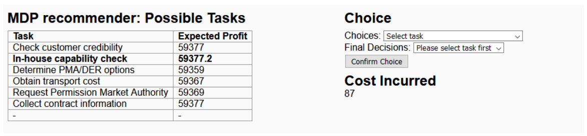
(a) Current state; tasks filled green and brown have been performed; tasks with green and brown lining are enabled



Information

Task	Variable	Value
Obtain Net Repair Cost	Net expected repair cost	1500
Obtain expected amount of repairs per year	Expected repairs per year	100
Check customer credibility	Customer creditworthiness	-
Estimate Current Market prices	Current mean market price	-
In-House Capability Check	In-house capability	-
Determine use of PMA/DER parts	PMA/DER possibilities	-
Determine Over and Aboves	Over and above options	0
Fix Contract Length	Contract period	3
Obtain transport cost	Transportation cost	-

(b) Case File



MDP recommender: Possible Tasks

Task	Expected Profit
Check customer credibility	59377
In-house capability check	**59377.2**
Determine PMA/DER options	59359
Obtain transport cost	59367
Request Permission Market Authority	59369
Collect contract information	59377
-	-

Choice

Choices: Select task
Final Decisions: Please select task first

Cost Incurred

87

(c) Recommendations

Figure 2.6: Recommender view

enabled. Because of constraints for this specific state, the final decision is not yet allowed and the decision maker can choose between six different tasks to continue the DIP. In expectation the optimal choice would be to perform In-house capability check. However, the five other tasks are also enabled in the current situation and their expected profit is also given. In this specific state most of the important information is already collected and therefore the differences in profit between the different options are marginal.

The recommender shows the allowed options and the expected profit for each decision in the *Possible Tasks* table in Figure 2.6(c). Moreover, because the MDP calculates the optimal future profit, the decision maker can estimate the effects of current decisions. Using the decision plan fragment that is inserted in the user plan, shown in Figure 2.6 with the brown color, the CMMN engine can

Table 2.5: Comparison MDP vs Decision Tree

Information	*MDP*	*Decision tree*
Expected profit	15867.6	8226.0
Number of final states	136	1254
Average retrieval costs	143.5	100.06
Expected revenue	16011.0	8326.06

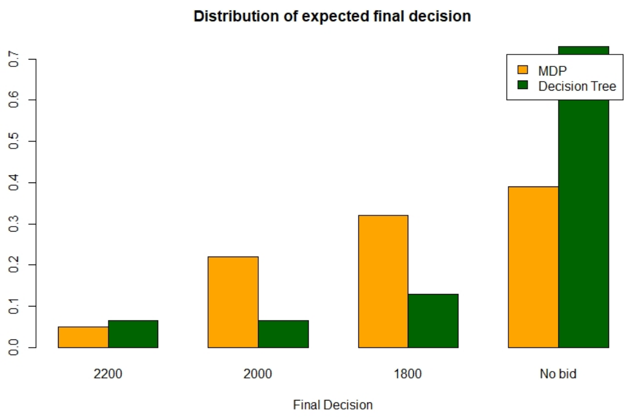
enact on the decisions that are taken. In accordance with Figure 2.1, the CMMN engine returns the new state of the process and calls for the MDP recommender in the new state. A demonstrator for this specific case can be accessed via `https://heshuis.github.io/optimizinginformationgatheringtool/`.

2.7.4 Numerical analysis

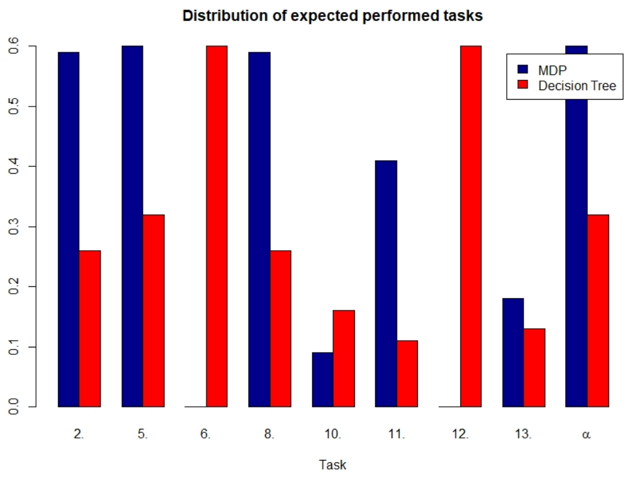
We compare the outcome of the approach with a decision tree that is based on the current decision process and is created in cooperation with sales managers in the company that inspired the example. The decision tree is shown in Figure 2.8 in appendix 2.A. In Table 2.5 we introduce some statistics based on the case model that was introduced in Section 2.3. Figure 2.7(a) shows the distribution of final decisions over all possible outcomes of the case file items. Figure 2.7(b) shows the proportion of cases in which a task is executed. Note that we removed the *required* tasks (3,4,7,9,β) as they are always executed to finish the process.

The main comparison can be made based on the expected profit of both methods. The MDP is able to double the expected profit compared to this human decision tree. If we look at the final decision distribution in Figure 2.7(a), clearly too many process instances are discarded in this decision tree. Also the distribution over the information sources that are acquired is, apart from the structurally enforced part, rather different. Furthermore, the decision tree also has a larger number of final states, meaning states where no more information is acquired. Together with the average retrieval costs, this implies that the decision tree is not complex enough to capture the dependencies between different attributes as defined in the information structure.

In Table 2.6, we show the results for nine randomly selected instances of the Fokker case process. Hence, we can show how both algorithms work for those instances. There are five instances where both algorithms conclude it is not profitable to bid for



(a) Decisions



(b) Tasks

Figure 2.7: Distributions

Instance	*Final Information State [A,..,I]*	*Profit MDP*	*Profit Decision tree*
1.	[2100, 40, 1, 1900, 1, 100, 0, 1, 150]	1656	-122
2.	[1500, 100, 1, 2100, 0, 100, 1, 4, 100]	69794	65469
3.	[2100, 80, 1, 2100, 1, 200, 0, 1, 50]	-45	-45
4.	[2100, 40, 0, 2300, 0, 200, 0, 3, 150]	16385	14692
5.	[1900, 100, 0, 2300, 0, 200, 0, 4, 200]	-45	-45
6.	[1900, 40, 0, 2300, 0, 200, 0, 3, 150]	-45	-45
7.	[1500, 40, 1, 1900, 1, 100, 0, 1, 150]	-45	-45
8.	[1700, 80, 1, 1700, 1, 100, 0, 2, 200]	27416	27419
9.	[1900, 60, 0, 1900, 0, 100, 1, 1, 150]	-45	-45

Table 2.6: Comparison of 9 instances

the maintenance contract. Therefore, they make a loss on the pieces of information that are always required to acquire. However, looking at instance 8, the human-built decision tree actually makes a better final decision than the MDP or makes that decision with less acquired information. This instance is a good example why the introduced approach does not only use the MDP algorithm but always requires the final opinion of the decision maker, where they can use tacit knowledge to judge the instance. However, including the instances 2 and 4, where the MDP is clearly outperforming the decision tree, we can see that on average for those 9 instances the MDP does improve upon the decision tree.

The level of complexity is high for a human decision maker using the decision tree. However, the number of nodes in this tree (14) is marginal compared to the number of states of the MDP, which depends mainly on the state space. The size of the state space can be estimated by taking the product of the number of all possible values for all tasks. Hence, for the example we have five information-acquiring tasks with five possible values, four information-acquiring tasks with three possible values and five procedural tasks/ stages that can either be performed or not, i.e., $5^5 \cdot 3^4 \cdot 2^5 = 8100000$ states. Note that the procedural constraints can cause a part of those states not to be explored during optimization. The introduction of new tasks will cause the state space to increase exponentially, making it computationally hard to find an optimal solution. For the example, optimizing the MDP took 49 minutes on a Intel i5-8365u CPU. In case of further scaling of the problem, one quickly runs into both memory and computation power problems. This specific problem of scalability is focused on in Chapter 3.

For this specific example the statistics have proven the value of using our approach during deployment with a doubling of the expected profit. Together with the steps that have been followed during the design time approach in Section 2.7.2, the approach has been shown to be feasible and applicable in real-life process cases. Furthermore, by using a real-world case and by validating the approach with decision makers, we show that this approach is improving upon the current approach that is based on decision trees. Given the necessary constraints, we consider this approach valuable for many more DIPs in different areas of expertise, such as financial services or governmental tender procedures.

2.8. Related Work

Process modeling languages like BPMN and UML Activity Diagrams offer basic support for modeling decision points. To enable reuse of such decision logic, the modeling language DMN was developed to declare recurring decision points in DMN decision tables that can be referenced in tasks of business processes [Object Management Group, 2020]. DMN focuses on what information is needed to make decisions. The last few years, however, several authors have recognized that decisions often are an integrated part of the business process that actually even controls the flow [Mertens et al., 2017, Yousfi et al., 2015, Hasić et al., 2018]. But there is no guidance for decision makers about which information to gather in order to perform DIPs both efficiently and effectively.

Hasić et al. [2018] introduce five necessary *principles* to connect a process model and a decision model: The first principle states that the decision logic should be externalized as a service to the process. In this chapter, we use a separate MDP tool that implements the decision logic, which could be realized as a service. A second principle is the need to model all data objects. Here, we use case file items, modeled as data attributes. The third principle of always modeling all possible decision outputs is followed in our approach. The fourth principle asks for ordering decision activities. By using a declarative language such as CMMN we allow the flexibility of deciding on the activity order and at the same time include any process constraints that define an order. The final principle involves making subdecisions. This specific extension of subdecisions is discussed in Section 2.9.

DIPs have been modeled using declarative process modeling. Vaculín et al. [2011] propose to model DIPs with declarative business artifacts. They develop design pattern support for the execution of DIPs. Guidance is modeled in the form of expert-defined constraints. Mertens et al. [2017] develop DeciClare, a declarative process modeling language targeted for DIPs. The process perspective in DeciClare is modeled with Declare, the decision perspective with DMN. No guidance support is provided. The approach defined in this chapter can be used in combination with DeciClare, replacing DMN-related parts with the decision fragments defined in Sections 2.6 and 2.7.2.

In general, the approach can be used in combination with other declarative process modeling languages, notably DCR graphs [Hildebrandt et al., 2011, Debois et al., 2014] and Declare [Pesic et al., 2007]. For instance, in the context of Declare, recommendation support has been developed, based on past process executions and their costs [Schonenberg et al., 2008]. The approach in this chapter is much more advanced, since it considers optimized decision making based on actual output data of tasks.

Next, there are approaches to guide the execution of information-intensive business processes, which produce information products [Vanderfeesten et al., 2011, van der Aa et al., 2015]. The structure of the information products is specified in product data models, which are tree structures that declaratively define how a data item is assembled from other data items. Different strategies exist to optimize the assembly of the informational products. Our approach focuses on optimizing decision making within the execution of DIPs. Therefore, we use information structures, which focus on the value of data items for decision making rather than their structure.

In earlier work [Voorberg et al., 2019, Eshuis, 2018] the value of using MDPs to support the execution of DIPs that are modeled in an artifact-centric process language is already discussed. There we presented a detailed translation of Guard-Stage-Milestone process models into MDPs [Voorberg et al., 2019]. The present approach builds upon this translation by defining an approach that shows how DIPs expressed in CMMN can be configured such that decision makers are guided during execution of a process to optimize decision making.

AI planning has been used to support the execution of knowledge-intensive processes [Venero et al., 2020, Marrella et al., 2017, Sid et al., 2019]. A plan is a sequence of tasks such that a goal state is reached. Performing a task results in a new state. Tasks can be modeled at different levels of granularity. If the user deviates from the plan and ends up in a state that is not in the plan, a new plan is generated. While these approaches support the flexible ad-hoc execution of processes, they do not support decision making, especially making the trade-off between gathering more information and making final decisions.

For expert systems, several optimization approaches for information acquisition strategies have been proposed [Mookerjee and Mannino, 1997a], for instance using dynamic programming [Dos Santos and Mookerjee, 1993]. However, none of these consider the relation with process models, such as CMMN and DIPs consisting of information acquiring tasks, procedural tasks, additional tasks, and decision-making tasks. Finally, we use the notion of an information structure which links the benefits and costs of gathering data for final decisions.

In sum, the main contribution of this chapter is a flexible approach, based on optimization, for guiding decision makers in performing DIPs both efficiently and effectively.

2.9. Discussion

An important assumption for the CMMN user model is the use of one milestone, which concludes the process. However, milestones are part of the CMMN language as an opportunity of setting subgoals during the process. Related, CMMN allows to create hierarchy in a DIP. By using substages one can define different layers of tasks and decisions to meet subgoals. This structuring can actually be of great value to larger DIPs. By introducing more milestones and stages, one could create a hierarchy of smaller size decisions. This decreases the state space of the DIPs dramatically. For example, if we cut a 10 attribute process all with a domain of four values into two smaller size processes, we can go from as much as 1048576 states to 2048 states. We use this idea to build a new approach in Chapter 3.

The existence of an MDP solution relies on the availability of probabilities and the assumption that those probabilities represent reality. Although frequencies of historical cases can give a good estimate, we know that in real-life things are always different. This is why we introduce the MDP as an advisor to the decision maker. Tacit knowledge can recognize specific historical cases and see dependencies between those and the current process. Still, we show in Section 2.7.4 that in expectation the MDP has a higher profit. This does not rule out that for specific cases the decision maker can use additional information to create an even higher outcome by diverging from the 'average' advice. However, as back bone for the complete set of cases, this MDP model is shown to make a real difference.

An alternative approach to MDPs are attribution models. Using Shapley values or the removal effect in the Markov model, one could try to find the marginal contribution of each individual attribute. However, the dynamic adaptation to newly available information cannot be used in these techniques. This is a crucial part of our solution, as it allows us to make better final decisions at the right moment.

2.10. Conclusion

This chapter introduced a new approach for decision support for information gathering in Decision Intensive Processes. The approach translates a CMMN process model into an MDP. Using the optimal solution for the MDP model, a decision maker is given advice what information to gather in order to make a final decision that is both efficient and effective. The approach introduces a new CMMN plan fragment to let a decision maker use MDP advice to control the process execution. Using the approach, decision makers performing DIPs are supported in deciding what information to gather to optimize the final decisions. Although these information-gathering decisions could be automated, we value the opinion of the decision maker and view its role as essential to decide which information to gather for a specific case.

An exemplary case based on a real-world industrial scenario showed the feasibility of the approach. For this real-world scenario, we compared a basic implementation of the approach using an MDP optimizer with a decision tree that was constructed by a decision maker from industry. We showed that the algorithm can double the expected profit compared to the decision tree. We also showed that there are still instances where the decision maker can make better decisions, e.g., based on information that is not available to the model, which suggests that a human decision maker should be in the lead.

Appendix

2.A. Decision tree

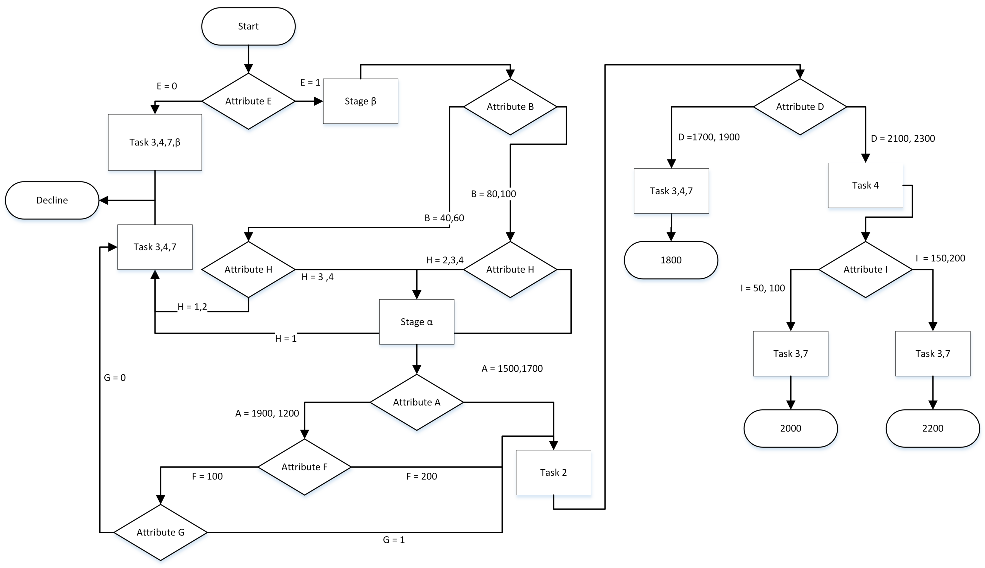


Figure 2.8: Decision tree

3
Hierarchical decision making in Decision-Intensive Processes

3.1. Introduction

Optimizing DIPs with a single decision is difficult. However, optimizing a DIP where multiple decision-making tasks are included and dependent on each other, is even more complex. The interplay between information gathering, information interpretation and decision making is a struggle for many decision makers. The complexity of many DIPs often comes from the fact that it involves many subdecisions that map as input to other decisions. Hence, optimizing a policy on how to run such a large DIP is either very time-consuming or even impossible. In Chapter 2 a first approach has been already introduced for small DIPs. However, this approach is not powerful enough to solve complex processes. The biggest problem lies in the scalability. Introducing extra tasks and information in the DIP increases the computation time exponentially. Therefore, there is a need for an approach that is also scalable to large size DIPs where finding a strong policy is more complex. Henceforth, we will refer to such large-scale DIPs as complex DIPs.

Every year companies are able to collect more information and to remain competitive it becomes inevitable to make use of this information. The sheer size of data sets already makes it complex to filter out the right information in many cases. At the same time, we can see that companies that deal the best with this information position have become the new giants of the world. Companies that want to keep working on optimizing decisions and DIPs, need solutions that can handle these vast amounts of information and the corresponding complex DIPs.

In literature, the need to combine process modeling and decision modeling has been acknowledged already multiple times. Both models have their respective standard in BPMN [Object Management Group, 2010] and DMN [Object Management Group, 2020]. Hasić et al. [2018] introduced a five principle idea that was necessary to integrate process and decision models. Also, Bazhenova et al. [2019] show that it is possible to even derive DMN models from BPMN models if the process has completely been specified and van der Aa et al. [2015] shows an opposite idea where a BPMN model is derived using a Product Data Model. However, in this chapter we do not only combine the process and decision modeling, but we also include an important data view. The collection of data does impact the process flow and the decision flow and, hence, should be further integrated to optimize DIPs. The addition of optimizing processes causing impact due to new information during the process, has been initiated in Chapter 2.

It has been shown that one of the common ways for humans to simplify a complex problem is by dividing it into smaller subproblems [Sarafyazd and Jazayeri, 2019]. Similarly, also organizations have a feel for diminishing complexity by dividing a problem into smaller problems [Simon, 1981], often also referred to as divide and conquer algorithms [Sen and Kumar, 2019]. Each subproblem is solved with its own subdecisions after which the full problem can be solved using these hierarchically decomposed problems. Hence, subdecisions create hierarchy. A subdecision gives an outcome that can be used as an input to a higher decision process. An example of this is the hierarchical structuring that exists in many enterprises. The Chief Executive Officer is not responsible for making all the decisions, but is provided with an overview of lower level decisions and possibilities such that she can make the main decisions, e.g., the decision that needs to be taken on a large investment. The financial department is responsible for arranging the funding and looking carefully at the return from the investment. Meanwhile, the human resources department is responsible for checking if the current job market is capable of delivering the necessary employees. The sales department checks if the investment is appropriate for the market fluctuations that it expects on the longer term. Now the CEO is only responsible for bringing together all the advice of the departments and deciding based upon them.

We propose a new approach using hierarchy in decision-intensive processes to facilitate faster and more realistic decision making. Moreover, this allows us to solve much more complex DIPs. Due to the dynamic impact of new information, the decision hierarchy can not be deduced from the process view of the DIP. Instead, we use a Decision Requirement Graph (DRG) that is part of the Decision Modeling Notation (DMN) framework. This DRG shows the mutual dependencies between decision outcomes in a tree structure that perfectly allows us to use the hierarchy of decisions. Using this hierarchy, we can decompose the entire process into smaller subprocesses that function as independent *Optimizable DIPs* (ODIPs). The outcome of these decomposed ODIPs, however, is one of the input parameters of the next level in the hierarchy. As such, we can converge to a single main decision that is optimized. The main decision concerns which information parameters to collect and therefore decides which subprocesses are optimized to return their decision as information. Figure 3.1 shows an example of the decision hierarchy of the process according to the DRG notation (see Section 3.3.2). Next to the decision one can see the flows of the problem. During the design-time of the approach where the policy is optimized, the main decision DIP determines which attributes are required for the optimal policy. Hence, we have a top-down imposed policy that starts from the main decision. During run-time we start with the lowest level decision processes, since the outcome of the decisions are inputs for higher level process steps.

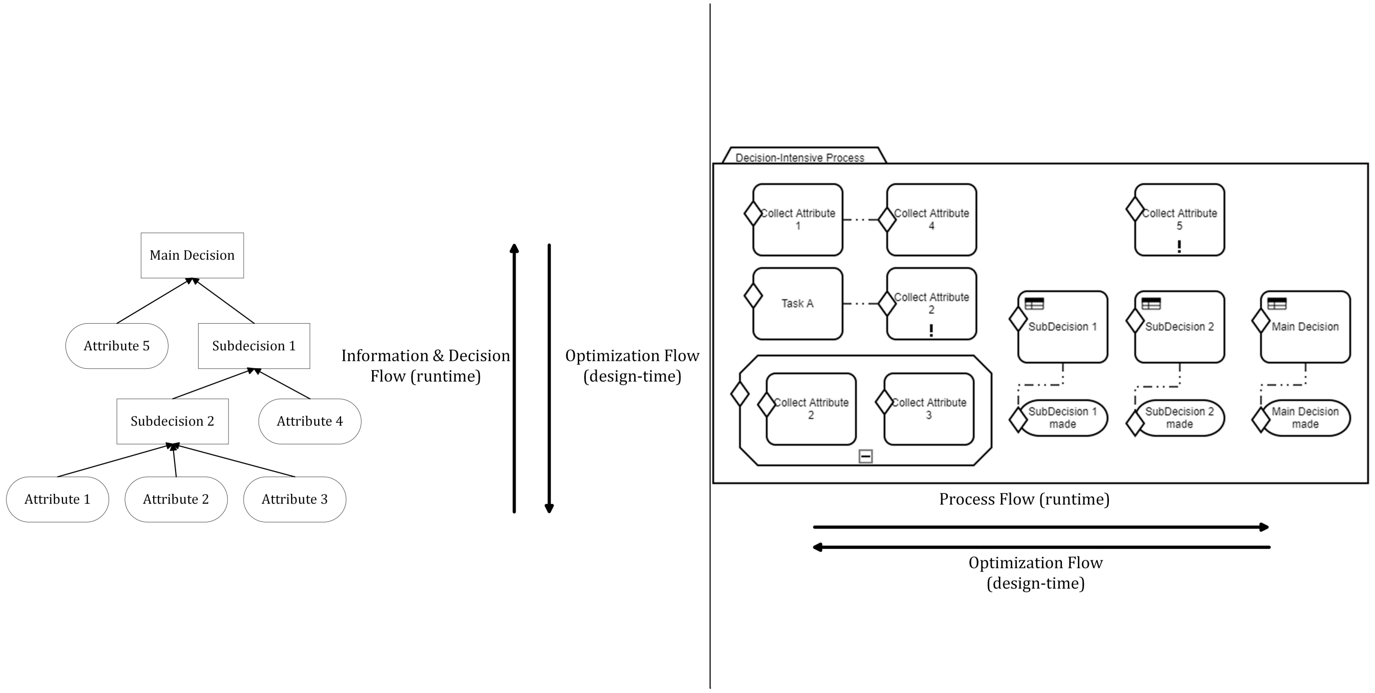


Figure 3.1: Overview of the approach

It is important to acknowledge the loss of optimality for this solution. When for the complete decision-process an information parameter is collected that is the outcome of a subprocess, we are required to finish that whole subprocess (optimally). However, because during the collection of that process all other options in alternative processes are not considered, we can not guarantee the optimality of the solution. Hence, we pay a price to solve these large scale problems. Note however that the full-size dynamic program that does take all subdecisions into account in one model is too large to be solved. Consequently, it is preferable to have a sub-optimal policy based on the available information than having no policy at all. Furthermore, since the optimization is performed from the top downwards, it can be that some of the subprocesses are not informative enough resulting in these complete subprocesses rendering obsolete for every problem instance. This can indeed result in both positive, since the subprocess does not need to be optimized, or negative consequences, rendering information useless.

In this chapter we will introduce the steps of the approach that will lead to the optimization of a complex DIP by decomposing it into independent ODIPs. We introduce a small example case for which we show how the approach works and we introduce a full scale complex DIP. Both are optimized and we study the numerical results that come out of this, both in terms of computation time and policy behaviour.

This chapter contains the following main contributions.

- We introduce a new view on how CMMN and DMN are complementary in the optimization of complex DIPs.
- We make use of natural hierarchy in complex DIPs to realize a decomposition into a set of small ODIPs.
- We show the effectiveness of this hierarchical decomposition in optimizing large and complex DIPs.

This chapter is organized as follows. Section 3.2 introduces an example case which we will use to explain the approach and which we will analyse in the results section. In Section 3.3, we introduce the approach that has been introduced to optimize an Optimizable DIP. Furthermore, we introduce the notation of DRGs as this will be the host notation for specifying the decision hierarchy. Section 3.4 introduces the main constructs that are being used in both CMMN and DMN and show how these two can overlap for a single approach. Section 3.5 discusses the assumptions that need to be introduced before the approach can be explained. Section 3.6 contains the introduction of our new modeling approach where we use a DRG to optimize a complex DIP by decomposing it into multiple ODIPs based on hierarchy. In Section 3.7, we will show the numerical computations with regards to cost optimization and time effectiveness using the approach on both the example case and a more complex DIP. In Section 3.8 and 3.9 we discuss the related work and any important assumptions, respectively. Finally, in Section 3.10 we conclude the chapter.

3.2. Example Case

In this section we will introduce a small example case that we will use in the further stages of this chapter to explain how the approach works. Note that the complexity of this problem is rather low, i.e., the case can still be solved with the dynamic approach for ODIPs. The main goal of this case is to explain the new approach and compare it to the previous approach.

Suppose that someone has the idea in mind to organize an outside party in the coming weekend. To make this party a success there is a number of variables that can impact the result and which should be checked before deciding on organizing this party, yes or no. For the party to be a success one would like a few close friends to be there. Furthermore, since the party will be outside the weather should be good enough to make it a success. Finally, there needs to be a caterer to take care of the food and so you have to see if the caterer is available. Regarding the close friends, there are three close friends that you would like to invite: Marc, Joey and Anne. Joey and Anne are a couple while Marc is still single. To be happy enough with close friends coming to the party, there are some specific rules. Since Marc is single and attractive he always makes a party more successful and so he counts for two. Also, when Joey and Anne are there together they are much more fun than just one of them coming alone. For Joey and Anne, it is hard to get a hold of them to ask if they are available since they are almost always together. So if you get a hold of one of the two, you are able to ask both for their availability. All three friends tell you that either they will not come, or they will surely come or they might come. (For the latter case, we assume a 50% probability that the might come will turn into a come.) For the weather, there are four important variables that determine how good the weather will be. First of all, both the expected % of rain and temperature are important. But, since you want to set up a tent you have to check the wind speeds also and finally it would be appreciated if the sun is really shining and no clouds are there. It is possible that either the weather is not good enough, or the weather is good enough but not amazing, or the weather will be amazing. Since the tent is borrowed from your dad he needed the promise to always check the expected wind speed before setting up the tent. In appendix 3.A one can find the exact way of modeling what is the weather type depending on these four data attributes.

The decision problem for this process is to decide whether or not to organize the party. The goal is to invest the least amount of time/cost in finding out all the information while not losing money on the organization of the party due to the party being a failure. The time investment in collecting information is $2 for all attributes. If the party is organized but in the end one of the things goes wrong, you lose $200. But if you do not organize the party while it would have been a success, you lose $50. A successful party earns, depending on good or amazing weather, $100 or $200 from tips of the visitors, respectively.

3.3. Preliminaries

3.3.1 Dynamic decision-support for optimizable DIPs

In this section we shortly introduce the approach that was built to give dynamic decision-support for specific Optimizable DIPs (ODIPs). The approach translates a DIP from the Case Modeling and Management Notation (CMMN) language, in which we model decision-intensive processes, into a Markov Decision Process optimization problem for which we can use known techniques to optimize the modeled processes. For further explanation on CMMN we refer to Chapter 2. The optimization consists of both the information acquisition phase and the decision-making phase. The approach introduces two new artifacts that are necessary for the deduction: An ODIP and an Information Structure. We will discuss both and explain how together they can be used to optimize decision-intensive processes.

Optimizable Decision-Intensive Process

An Optimizable Decision-Intensive Process is a subclass of a regular Decision-Intensive Process. The additional constraints that apply for an Optimizable DIP are:

- Tasks cannot be recurring, hence we have *stable information*.
- There is a *single milestone* that closes the ODIP.
- The result of each task is *instantaneously* known when performed.

The assumption of having stable information refers to the interdiction of redoing a task which could return different information. In that case, optimization is not possible since either a crucial information parameter will just be recollected until it returns the right value or the same value will be returned each time, making it useless. For our example case this means that once one of your friends gave you a response, the assumption is that the response will not change anymore. The constraint on milestones is that they can only be used to close a part of the process. Since an ODIP only contains tasks that directly impact the final decision of the process, we can conclude that subprocesses are not part of the design. Therefore, for an ODIP, only a single milestone can be used to close off the decision process. Finally, all tasks are assumed to finish instantaneous. This means that we do not take into account possible time delays between asking and retrieving information. This constraint can be relaxed such that timed tasks can be introduced. However, this still makes the problem too complex to allow us to solve the problem. In Chapter 5, we discuss the use of Reinforcement Learning to optimize large scale problems where the instantaneous assumption is relaxed.

Using these additional constraints, we can build a CMMN model that represents the example case of Section 3.2, which is shown in Figure 3.2. The information items belonging to the tasks can be found in Figure 3.3.

Information Structure

An Information Structure is a function that connects all the information attributes to an outcome which must be optimized, i.e., it allows to interpret and analyze the obtained information towards a decision. The input is a list of data attributes. Based

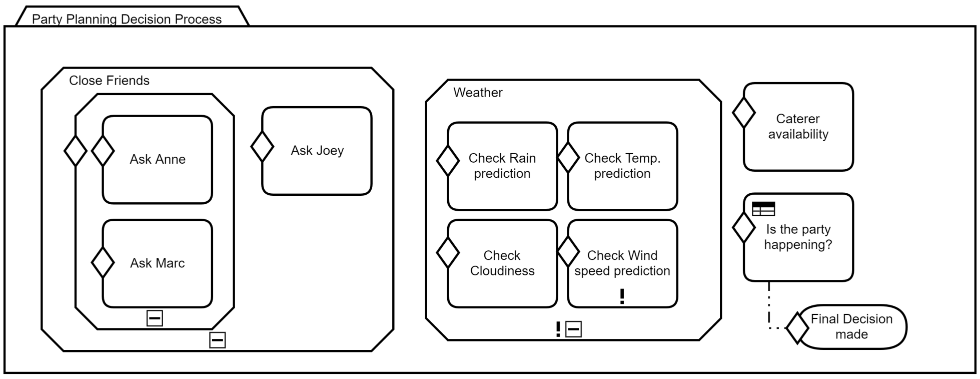


Figure 3.2: CMMN model of a party planning DIP. The corresponding information attributes are introduced in Figure 3.3.

on the attribute values and the decision, a reward is returned. The ODIP approach connects information to a process structure, while the information structure is used to connect information and decisions. Optimizing the information structure requires knowledge on the values of the included data attributes such that one can choose the best decision. Note that at the start of the process the attributes are uncertain, meaning that we do not know the exact value but we do know the possible outcomes and the corresponding probability distribution. For the example case, we can define the information structure based on the given information. It returns either a loss or a profit on the organization of the party, which depends on the number of close friends attending and the weather. In this chapter we assume that it is possible to define information structures per subdecision based on the available process set-up and information.

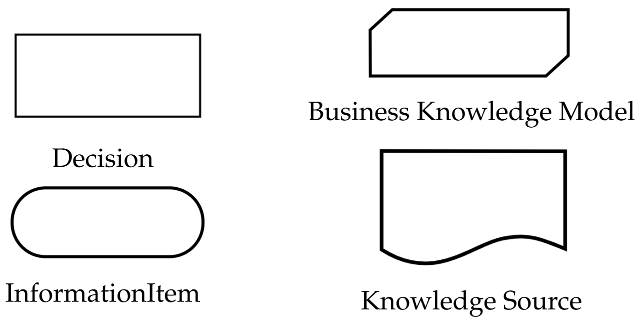
MDP optimization

An MDP is a sequential decision-making process for which we use a so called Markov chain that describes the probabilities of arriving in different states of the process. In our case, a state denotes the set of attributes and their (currently already) acquired values. The information structure connects these attributes into a value function that defines the rewards in the current state. By taking actions (tasks), the state can be changed, due to newly acquired attribute values, resulting in a changing reward. Since actions in the approach improve the information position due to new information attribute values, we have a better idea of the profits when making consequential decisions. Knowing the possible outcomes and corresponding probability distributions allows us to do a backward recursive search to find the optimal policy for relatively small instances of MDPs.

3.3.2 Decision Requirement Graph

A Decision Requirement Graph (DRG) is part of the DMN architecture according to the specification of OMG Object Management Group [2020]. A DMN model consists of two elements: the DRG and a decision table. The decision table can, however, also be replaced by a different decision-making alternative such as a function, a list of decision rules or informal text describing the decision rules. In our approach, decisions are based on the information structure, which is a function. Therefore, we focus on the DRG. The DRG models the domain of a decision. It shows the important elements that are involved in making decisions and their dependencies. The elements that are used are: Decisions, Business Knowledge Models, Knowledge Sources, Input data and a Decision Service. In Table 3.1, the respective objects are shown.

Table 3.1: DRG constructs



We need this DRG to model the relation between process steps and the multiple (sub) decisions that are taken in this hierarchical model. The structure of a DRG, see Figure 3.3 , consists of a *Decision* (rectangle) that requires a certain input. This input is denoted by connected arrows and can consist of information requirements, knowledge requirements, and/or authority requirements. The required information can be found in Input data elements (rounded rectangle) that function as a decision

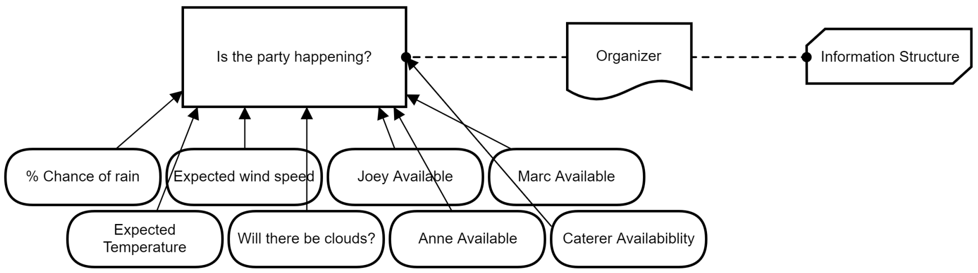


Figure 3.3: DRG representation of a party planning DIP.

parameter. The Business Knowledge Model is a function that encapsulates the decision knowledge and can fulfill the knowledge requirement (snipped rectangle). Finally, a knowledge source denotes an authority for a decision (waved rectangle). Together these three inputs allow for a decision to be made, i.e., these three inputs can be connected to a decision element. It is important to mention though that a decision can also function as an information requirement for another decision. In Figure 3.3, one can see all introduced parts of a DRG graph for the example case of Section 3.2.

3.4. Dynamic decision support for complex DIPs

If we return to the example case in Section 3.2 and how to optimize this DIP, one can approach this problem in two ways. Either we introduce the DIP as a single process that considers all involved attributes and we optimize based on all attributes in one model, i.e., the dynamic approach for ODIPs according to Section 3.3.1. This could result in a policy where you first call Marc, then check the predicted temperature after which you call Joey and Anne. Although it is possible to apply the dynamic approach for ODIPs on the case and find an optimal solution, this is mainly due to the simplicity of the example. In Section 3.7, we introduce a complex DIP for which this first option is not feasible.

The alternative is that we use the hierarchical structure based on subdecisions to decompose the problem, where a subdecision functions as input for higher level decisions. For both decisions on if the weather is good enough and if there will be enough close friends, you don't want to spend too much time and so you would like to find a smart order to gather the information. If one of the three variables friends, weather or caterer, is not good enough you prefer to cancel the party. In Figures 3.2 and 3.3 we have introduced the ODIP, according to the introduction in Chapter 2, and basic DRG of this problem. For the new approach where the DIP is decomposed, we optimize the problem per subdecision. This results in a policy that will, if necessary, first check if I have enough friends available, and only then check if the weather will be good enough.

In Section 3.6 we introduce the approach that one should use to decompose a complex DIP into a tree of Optimizable DIPs. Such a hierarchy consists of a set of decomposed ODIPs that can each be independently solved and where the outcome of an ODIP can map as the input for a higher level ODIP. An Optimizable DIP hierarchy consists of three complimentary parts: The CMMN model, the DRG model and the existing ODIP template. In the following subsections we will introduce the most important artifacts in both CMMN and a DRG that are crucial for an ODIP hierarchy. Note that we leave out some details of the ODIP as they are already discussed in the previous chapter.

3.4.1 Combining CMMN and DRG

In this section we focus on how CMMN and DRG notation overlap, creating a valuable combined approach that allows us to decompose an ODIP, modeled by a CMMN case, into multiple smaller ODIPs. Figure 3.4 shows the overlap of the two models in UML notation. The following paragraphs focus on the specific entities in this UML model that are crucial for the decomposition approach.

Decision-making task/Decision. A decision-making task in CMMN denotes a task that inputs information and outputs a decision. CMMN even has a specific notation that refers to the use of DMN for this. Similarly, in the DRG notation that is focused on decisions, there is the decision artifact (Table 3.1). Therefore, every decision-making task that can be found in the CMMN model of an ODIP, should also have a decision in the DRG. These decisions also appear in the approach for ODIPs as they map the final decision of the MDP that terminates the process.

In an ODIP, only one decision-making task is allowed to be available. However, for the ODIP hierarchy in this chapter we drop this constraint. There is no maximum

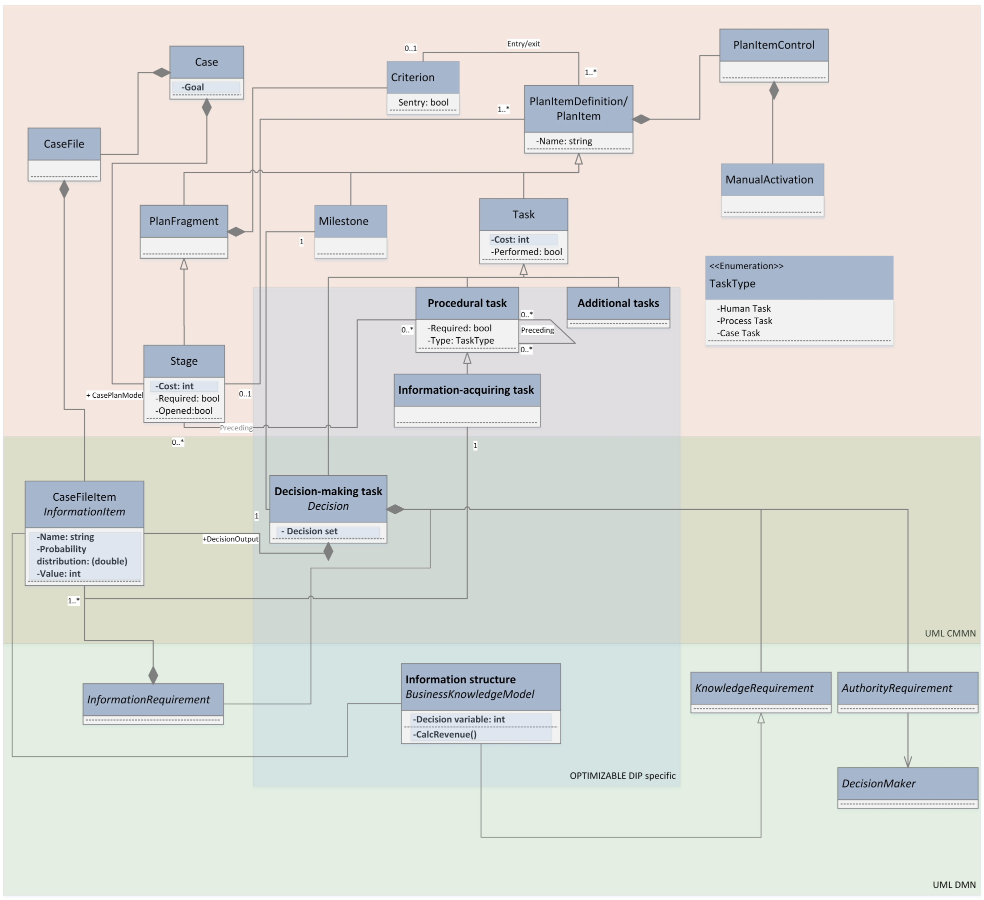


Figure 3.4: CMMN meta model (pink, normal), DMN meta model (green, italic) and Optimizable DIP environment (blue, bold)

for the number of decisions in the process. The decisions should all appear in the DRG though and should all be connected into one main decision. Each decision should also appear as a decision-making task in the CMMN. There is one important requirement that we require every task in the ODIP to appear in at least one decomposed ODIP.

CaseFileItem / InformationItem. Information is crucial to make good decisions. In CMMN we use CaseFileItems to denote which information attributes are connected to which case tasks. Such CaseFileItems have their own data view that can be modeled using for example UML notation. For DRG there is a similar concept called the InformationItem (Table 3.1). An InformationItem is used as an attribute that is needed to make a decision. It is important to mention that, opposite to the standard rules of DRG, in this approach we are not requiring all attribute values to be known to make a decision or to be able to make a decision. This is because the decision maker can choose to not collect all information attributes. To distinguish between known attribute values and unknown attribute values we introduce a special construct of the Information item element according to Figure 3.5. The decision is still optimized, but based on the expected value of the random variable distribution of the unknown attribute values.

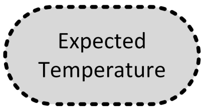


Figure 3.5: Uncertain Information item

Information Structure / Business Knowledge model. The introduced artifact of an Information Structure for the ODIP is an new concept in the DRG notation, but it can be seen as an example of a Business Knowledge model. The Business Knowledge model is a rather broad construct that represents knowledge in general often using a function based on the available information to come to a decision, which is why the Information Structure can be seen as such.

Decision maker / Knowledge Source. Already in the dynamic approach for ODIPs we acknowledged the value of a decision maker who keeps in control of the process, backing up the optimizer and sometimes changing decisions based on intuition or inside information. In the DRG notation such objects are also recognized and denoted Knowledge Sources. Although there are more objects that can be seen as a Knowledge Source, we assume it to always be the decision maker. The decision maker is the responsible authority that approves the optimized policy following from our MDP.

Knowledge Requirement & Authority Requirement & Information Requirement. Although the above described concepts are necessary to build a DRG, there is a specific focus on the functionality of these concepts. Because, while in general there seems to be a clear build up of a decision being made using a Business Knowledge model based on the available InformationItems, such concepts can also take multiple roles in the DRG. These roles are expressed in three different ways: Knowledge requirement, Authority requirement and Information requirement. For example, the outcome of a Decision can work perfectly as an Information requirement for a hierarchically higher decision. Similarly, a decision maker can be an authority requirement for a decision but a Business Knowledge model can also function as a knowledge requirement to the authority requirement.

3.5. Modeling assumptions

3.5.1 Process rules

An important part of the decomposition approach is the implementation of process rules in the constraints of the optimization method. For example, we allow the use of stages, required tasks and preceding tasks. While performing the decomposition, such rules can still be implemented. Any preceding tasks or stages that are not involved in the decision according to the DRG will be still included due to the CMMN model such that the constraints are still there. Similarly for any required tasks, we will impose the requirement on the involved decomposed ODIP. In the next section we will discuss how we extend the notation of an ODIP to use it for the decomposition approach.

3.5.2 Notation

If we look at the notation used to filter the MDP optimization problem from the CMMN model for an ODIP, there are a few comments to make. To distinguish between the different decomposed subprocesses, we use a notation according to the level in tree hierarchy and the number of subprocesses at that level. For example, the main decision process has id: $\mathfrak{D}(0.0)$, while a second subdecision at the third level has level $\mathfrak{D}(3.2)$. Subsequently, for all introduced notation, a similar subset can be created: $T_{(0.0)} \subset T$, where $T_{(0.0)}$ denotes the set of tasks that are directly involved in the $\mathfrak{D}(0.0)$ main decision and T denotes the complete set of tasks. Any stages that contain a directly involved task are also part of the subprocess. For any subprocess $\mathfrak{D}(\cdot,\cdot)$, the complimentary decision-making task is denoted by $d(\cdot,\cdot)$. Because of the decomposition, the outcome of a lower level decision is an information attribute for a next level. Hence, for every final decision $\mathbf{a} = f \in \mathcal{F}$ of a subprocess, there is a fraction of the cases in which this final decision is taken. Therefore, we define $X_{(\text{level,number})}$ as the random variable that represents the distribution over the set of possible final decisions, where the domain of that variable $dom(X_{(\text{level,number})}) = \mathcal{F}^{\text{level,number}}$. Furthermore, $c_{(\text{level,number})}$ denotes the expected costs for finding the true value of this variable, i.e., deciding on the subproblem. For each subproblem, the corresponding information acquisition tasks of all involved data attributes in the set D are connected to the DMN subdecision $\mathfrak{D}(\cdot,\cdot)$ and any procedural tasks according to the definition of Chapter 2. The remaining set of the tasks in T can be seen as additional tasks to the specific process and will be discarded for this specific decomposed ODIP. Finally, for the n data attributes that are connected to a subprocess $\mathfrak{D}(\cdot,\cdot)$ and the costs of collecting these data attributes, we use the notation of $X^i_{(\text{level,number})}$, and $c^i_{(\text{level,number})}$, $1 \leq i \leq n$, respectively.

3.5.3 Repeated Information

Although information attributes can only be collected once, there is the possibility of multiple decomposed decisions making use of the same attribute. In such cases, if the attribute has already been collected in a fraction of the cases, we assume only the remaining fraction of cases can be contributed in the higher hierarchy. This is implemented by multiplying the acquisition costs with the remaining fraction of cases in which the attribute is not collected. For example, the optimization of a subprocess results in the collection of the attribute value in 50% of the scenarios. A hierarchically higher process uses the same attribute but in only 50% of the scenarios it still has to pay the costs of collection, since for the other 50%, the attribute has already been collected in the preceding subprocess. Hence, we impose 0.5 of the original costs when optimizing. For a subprocess acquisition of data attribute $X^i_{(\text{level,number})}$ in only 50% of the cases, we impose an expected costs of $0.5c^i_{(\text{level,number})}$ for the subsequent subprocess with the same data attribute. Since we do not track the exact set of cases for which the attribute is collected, it is not possible to know if the two subprocess complement each other to 100% of the scenarios. Therefore, we continue to apply the same procedure for the subsequent subprocess.

3.5.4 Procedural constraints

Next to the different tasks that are identified according to the UML diagram, there can also be additional procedural constraints. We distinguish both tasks that are required for the process and tasks that have other preceding tasks. For any required tasks, this will mean that the required constraint is pushed up through the decision tree since every higher subprocess will have to be required also. Any preceding task is seen as a procedural task that should always preceed the current task. Therefore, the preceding task is part of the decomposed ODIP of the consequential task.

3.6. Decomposition and optimization approach

In this section we introduce the approach that can be used to decompose a complex DIP into a set of ODIPs. Furthermore, we discuss the need of extending the CMMN model of the process with a process control fragment that keeps control of the process. Finally, we show how the approach works for the example case.

3.6.1 Approach

The decomposition and optimization approach looks as follows:

1. To decompose a complex DIP, one needs both the process/data view of a CMMN model and a data/decisions view of a DRG graph. Subsequently, each information item in a DRG must have a task in CMMN where the item can be acquired and each decision in DRG must have a decision-making task in CMMN. Moreover, all decisions in the DRG should have a hierarchy that leads to a single main decision (ranked according to Section 3.5.2). If these conditions are met, one can continue.

2. Take the lowest ranked subdecision. List the set of attributes that are information items to this subdecision including any possible lower finished subdecisions.

3. Retrieve from the CMMN model the information-acquiring tasks connected to the listed attributes and also any procedural or additional tasks that are

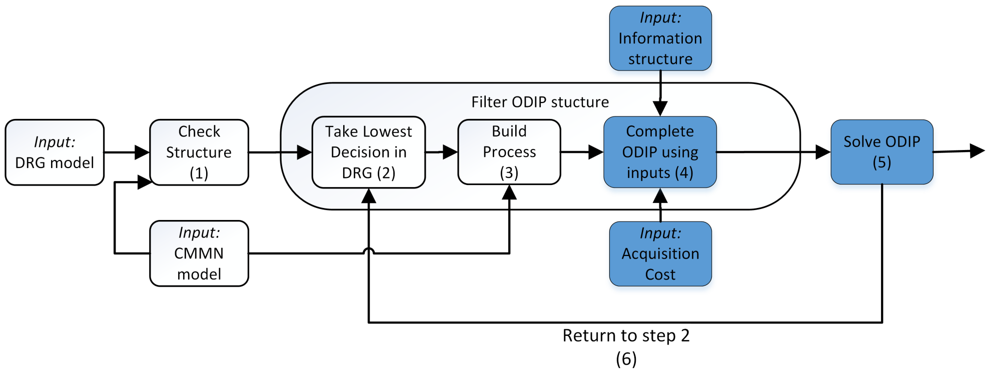


Figure 3.6: Schematic representation of the decomposition and optimization approach. The numbers in between brackets denote the different steps that are introduced in the approach. The blue nodes are imported from the ODIP approach in Chapter 2.

involved to perform these information-acquiring tasks.

4. Retrieve the necessary acquisition costs for each task, considering also earlier paid expected costs for repetitive tasks (See Section 3.5.3). Retrieve the Information Structure that is connected to this subdecision according to the DRG also containing the decision alternatives.

5. Optimize the ODIP that can be built based on steps 2,3 and 4 and results in a list of expected costs per collected data attribute, the cumulative expected acquisition costs and a distribution over the possible final decisions. Note that this information together allows to interpret this subdecision in the DRG model now as an information attribute for the higher decision problem.

6. If there are subdecisions remaining in the DRG for which the subprocess has not been optimized, return to step 2. Otherwise, stop.

A few remarks can still be made for this approach. First, the information structure is assumed to be defined per subprocess. Considering the independent targets that departments often have in companies, these information structures should be possible to derive. However, in future work we also aim at finding a method or approach that is able to decompose a main information structure into a multiple of them.

Second, when all ODIPs in the hierarchy have been optimized, one can call the optimal solution of the specific ODIP to find out what is the optimal policy to follow. If this policy requires the outcome of information attributes that are the outcome of a subprocess, it implies that this subprocess should also be performed according to the optimally derived policy. Even though a policy has been computed now, this does not mean that the process is controlled by this policy now. The control of the process must be handled by the policy in cooperation with the human decision maker. How to do this has been described in Section 2.6 in Chapter 2. The exact same approach can be used for this adapted approach, keeping in mind that the optimal action is based on the subprocess optimization.

3.6.2 Decomposition of example case

Considering the complete set-up of having both a CMMN model containing multiple decision tasks and milestones, and simultaneously having a DRG that models the decomposition of the different decisions, we conclude that there is the possibility of decomposing a complex DIP into subprocesses that each map as an independent Optimizable DIP. For each of these ODIPs it is possible to have all the necessary ingredients to deduce the optimization problem. This results in a policy which adheres to the constraints from both the process side and decision side of the problem. To further show the practical consequences and implications, we continue based on the exemplary case that was introduced in Section 2.3 where we will show how this process is decomposed and solved using two sub-processes under control of the main process.

1. Set-up DRG In Figure 3.3, we introduced a DRG model that modeled the decision requirements for the example case. However, this DRG did not contain the option of any subdecisions. Figure 3.7 introduces a DRG for the same process but then containing two subdecisions based on the description. According to this DRG there are two subdecisions: 1. Is the weather good enough? 2. Will there be enough friends? Note that these subdecisions can also be modeled as part of the process in the CMMN model as decision-making tasks (Figure 3.8).

2. Take the lowest rank subdecision Both subdecisions are directly linked to a single main decision and therefore the choice is free on which subdecision to start.

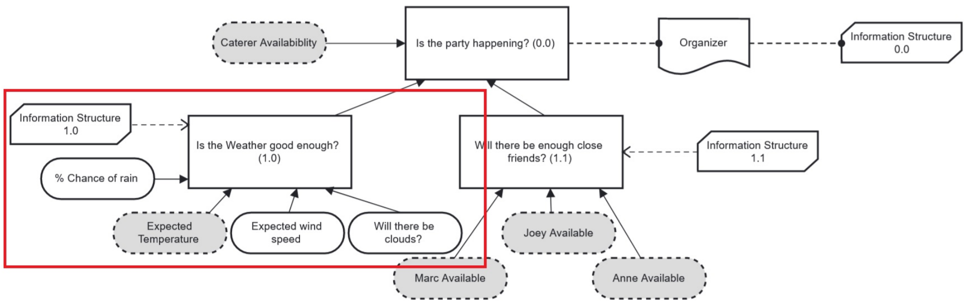

Caterer Availabiblity
Is the party happening? (0.0)
Organizer
Information Structure 0.0
Information Structure 1.0
Is the Weather good enough? (1.0)
% Chance of rain
Expected Temperature
Expected wind speed
Will there be clouds?
Will there be enough close friends? (1.1)
Information Structure 1.1
Joey Available
Marc Available
Anne Available

Figure 3.7: Decomposed DMN representation of a party planning DIP

We start with the weather subdecision.

3. Retrieve the involved tasks Based on Figures 3.7 and 3.8, we can conclude that there are four tasks that can collect four different information attributes for the weather subdecision. Furthermore, the wind speed prediction is a required task according to the process model in Figure 3.8.

4. Retrieve the costs and information structure The specific costs for this subprocess are defined according to the costs for the main process, a cost of 2 dollar per attribute. The information structure classifies the weather based on a decision tree into either bad, good or amazing. The profits for making a right or wrong choice are chosen such that the attributes are always collected when useful.

5. Optimize the ODIP In Figure 3.8 we show the decomposed processes in the CMMN model, where the red outlined process is now the subprocess on if the weather is good enough. The outcome of the optimization of this process based on all collected input parameters can be found in Tables 3.2 and 3.3, which are discussed in Section 3.7.1.

Since we have a solution for the subprocess on the weather, this can now be returned as an input in the DRG model. In Figure 3.9, the weather subdecision has now changed. Some of the attributes have been acquired and therefore now have a known value. Also, in the CMMN the decision-making task on weather has been performed and hence the outcome can now map as an information attribute to the next decision

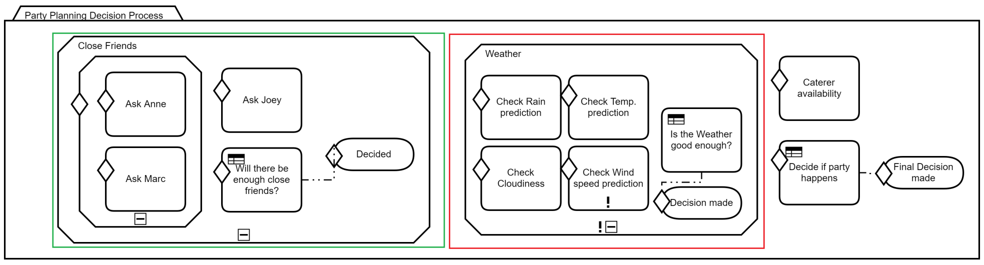


Figure 3.8: Decomposed CMMN representation of a party planning DIP

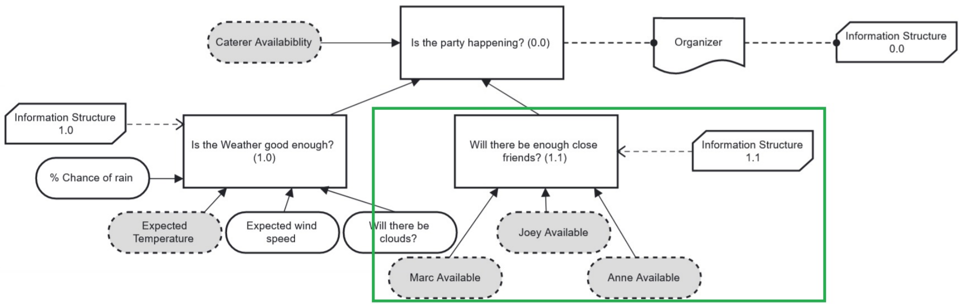

Caterer Availabiblity
Is the party happening? (0.0)
Organizer
Information Structure 0.0
Information Structure 1.0
Is the Weather good enough? (1.0)
% Chance of rain
Expected Temperature
Expected wind speed
Will there be clouds?
Will there be enough close friends? (1.1)
Information Structure 1.1
Joey Available
Marc Available
Anne Available

Figure 3.9: Decomposed DMN representation of party planning DIP after $\mathfrak{D}(1.0)$ has been optimized

in the hierarchy. Note that the wind speed prediction task was required in Figure 3.8 and hence the attribute has a known value in Figure 3.9.

6. While decision processes remain, return to step 2 Where the subprocess $\mathfrak{D}(1.0)$ has now been optimized, we return to step 2 to perform the same for the friends subprocess.

3.7. Results

In the results section we will discuss the numerical outcomes for the example case. Since the example case is small, we can just solve it optimally using an MDP. Therefore, we can compare the result of the approach with the optimal policy. Next to the small case, we introduce a full scale complex DIP and discuss the performance of the approach for this DIP.

3.7.1 Example case

After applying the decomposition approach, we can optimize the different ODIPs that together build a policy for the complex DIP. Moreover, because of the small size of this DIP we are able to also do an optimization based on a single ODIP such that we can benchmark the decomposed policy.

Policy comparison In Table 3.2 one can see the difference between the two policies. Policy A denotes the decomposed process where subprocesses are separately optimized. Policy B denotes the optimal process that optimizes the full process. The fractions in between brackets are the truly applied fractions adapted for higher level hierarchy decisions. We can see for the 'Friends' subprocess that the focus is more on Marc compared to the optimal solution. Furthermore, both solutions begin looking for information in this subprocess in 70% of the cases. This has to do with the 0.7 probability of the caterer being available. Furthermore, the optimal solution always checks for the wind speed. Apparently only the wind speed plays a crucial role in the weather prediction. This has to do with the requiredness of the specific task, which is why also the decomposed algorithm always does the weather prediction.

Table 3.2: Fraction of cases in which information is acquired. A is the decomposed process, B is the optimal process.

Subprocess	***Task/Stage***	A	B	Costs
Invite Close Friends ($\mathfrak{D}(1.1)$)	Contact Joey + Anne	*0.64 (0,448)*	*0.7*	10
	Ask Marc	*1 (0,7)*	*0.35*	8
	Ask Joey	*0.6 (0,42)*	*0.21*	2
	Ask Anne	*0.4 (0,28)*	*0.09*	3
Check Weather ($\mathfrak{D}(1.0)$)	Check % Chance of rain	*0.5 (0,5)*	*0*	2
Prediction	Check Expected Temperature	*1 (1)*	*0*	2
	Check Expected wind speed	*1 (1)*	*1*	2
	Check Cloudiness	*0*	*0*	2
Final Decision ($\mathfrak{D}(0.0)$)	Invite Close Friends	*0.7*	-	-
	Check Weather Prediction	*1*	-	-
	Check Caterer Available	*1*	*1*	4

Cost comparison In Table 3.3, we show the ODIP profit and cost for collecting information for the decomposed process, which can be compared to the result of the optimal policy in Table 3.4. The cost in between brackets denote the true cost given that the specific subprocess is only performed in a certain fraction of all cases according to Table 3.2 We see that the expected profit of the decomposed solution is higher than the optimal solution. Since the decomposed process is suboptimal this can be easily explained.

Table 3.3: Policy of decomposed DIP example case

Subprocess	**Outcome**			**Costs**	**Profit**	**States**	**Time (sec.)**
Invite Close Friends	Enough 0.167	Not enough 0.833		16.8 (11.76)	-	128	0.018
Check Weather Prediction	Amazing 0.777	Good 0.197	Bad 0.026	5 (5)	-	625	0.031
Final Decision	Party 0.12	No Party 0.88		17.2	3.1	45	0.007

Table 3.4: Policy of ODIP example case

Process	**Outcome**		**Costs**	**Profit**	**States**	**Time (sec.)**
Final Decision	Party 0.18	No Party 0.82	16.5	1.7	1171875	11492.73

Calculation comparison Tables 3.3 and 3.4 also contain the required time to solve the MDP of the ODIP. This is where the decomposed solution show its value. The complete MDP consists of more than one million states, resulting in a calculation time of 3 hours and 12 minutes. At the same time, the decomposed problem has three subproblems consisting of maximally 625 states that are all solved within miliseconds. Moreover, any independent decomposed processes at the same level can be optimized in parallel. For example, in the example case we can optimize $\mathfrak{D}(1.0)$ and $\mathfrak{D}(1.1)$ at the same time. The result is never optimal at the same level but saves a significant amount of time for this small case already.

3.7.2 Main case

In this section we introduce a case that is used to show the value of our new hierarchical approach. We have used the 'Custom bikes' from Garcia [2011] as

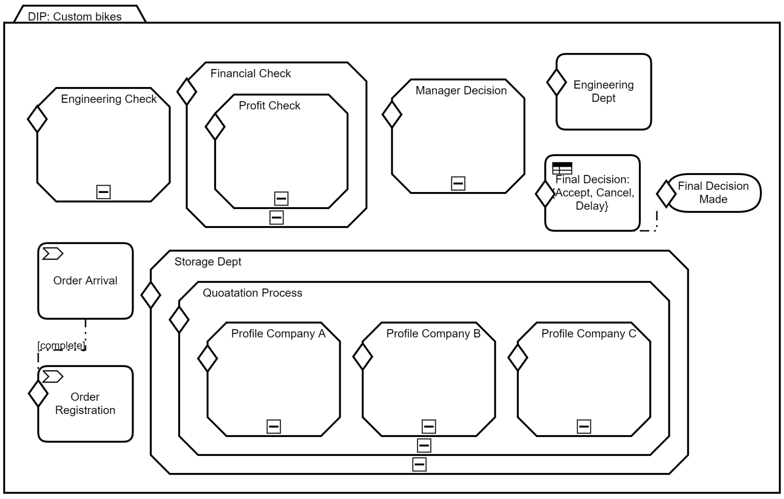


Figure 3.10: Summarized CMMN representation of the 'Custom bikes' case by Garcia [2011].

inspiration, resulting in a case description. The case relies on the main idea of a company where multiple departments are involved. In this case, the company has to reach a decision on whether the bid they offered for producing a custom bike is profitable and if they have the capabilities to produce the bike. In the end, this results in a decision to either accept, delay or cancel the order. There are multiple responsible departments in the company that have a say in what the final decision should be.

The full description of the case can be found in Appendix A. Figure 3.14 in appendix B shows the CMMN model that defines the full DIP, Figure 3.10 shows a small representation of it. Figure 3.13 in appendix B shows the DRG that belongs to the same process, Figure 3.11 shows the small representation of it. The information structures for each of the subprocesses have been derived based on the description in Appendix A. Note that for this process there is no actual profits or losses defined for wrong decisions. The costs for making wrong decisions have been set so high in the respective subprocesses that the resulting policy will invest the maximum amount of money in acquiring information. Given this, we will not define all information structures that have been used for the respective subprocesses. Tables 3.8 and 3.9 define for each subprocess the tasks and subprocesses that can acquire an attribute and for which costs this can happen. Furthermore, in Table 3.10, the probability

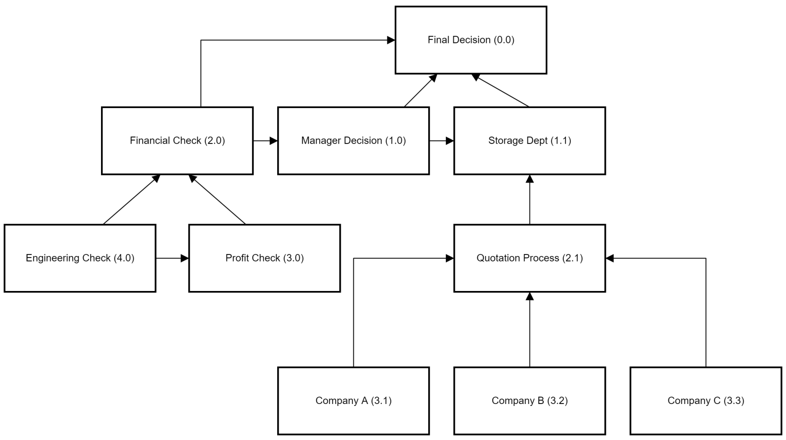


Figure 3.11: Summarized DRG representation of the 'Custom bikes' case by Garcia [2011].

distributions for all attributes have been defined.

General results Based on the introduced information in Tables 3.8, 3.9 and 3.10 (Appendix 3.E), and using the decomposition of the DRG in Figure 3.13, it is possible to have the set of decomposed ODIPs that can be optimized. Table 3.11 shows for each of these optimized ODIPs the outcome. The outcome consists of a probability distribution over the decision alternatives for the subdecision and the expected costs that can be accounted for finding the true value in this distribution. In Table 3.12, we show per subprocess what is the fraction of cases for which the specific task is performed acquiring its corresponding information attribute.

This process, in the end, has an expected cost to find the optimal final decision of 27.47. 57% of the offers that were made can be accepted, of which around 30% will have a delay. The most expensive subprocess is expected to be the quotation process where different companies have to be approached for making some specific parts of the bike. However, due to these high costs, this subprocess is only started in 49% of the cases. Especially in Table 3.12 one can see that for most information attributes

Comparison to Optimality For the *Financial Check* subprocess, it is possible to benchmark against the optimal policy where the process is not further decomposed. Table 3.5 shows the performance of the two alternatives. The optimal solution indicates a slightly higher profit, based also on collecting a bit more information. For the decomposed solution, the solution is rather close to the optimal solution also looking at the fraction of cases in which attributes are collected, shown in Table 3.6. The fraction in between brackets of this table denote the hierarchically true value incorporating higher level subprocesses that are already performing lower level processes in a fraction of the cases. For larger size problems we cannot benchmark against the optimal solution, but for the two small size problems that are studied in this chapter, the performance is good.

Table 3.5: Result of the Decomposed (DEC) versus Optimal (OPT) policy regarding the *Financial Check* subprocess.

Process	**Outcome**			**Costs**	**Profit**	**States**
	OK	*Medium*	*Not OK*			
Final Decision (OPT)	0.21	0.21	0.58	7.9	-7.4	78125
Final Decision (DEC)	0.15	0.18	0.67	7.7	-7.7	

Table 3.6: Fraction of cases for which the Decomposed (DEC) versus Optimal (OPT) policy performs the specific tasks regarding the *Financial Check* subprocess.

Subprocess	***Task/Stage/Subprocess***	Fractions DEC	Fractions OPT
Engineering Check	Determine # of custom parts	1 (0.67)	*0.67*
	Determine gearing type	0.074 (0.05)	*0.03*
	Determine frame type	0.0625 (0.04)	*0.04*
	Determine brake system	0.4375 (0.29)	*0.22*
	Determine steering system	1 (0.67)	*0.51*
Calculate Revenue	Determine # of custom parts	1 (0.22)	-
	Collect History from Customer File	1 (1)	*1*
	Determine Offered Price to customer	0.08 (0.08)	*0.17*
	Engineering Check	0.67 (0.67)	-
Financial Check	Engineering Check	0.24 (0.24)	-
	Calculate Revenue	1 (1)	-

Complexity of the solution We would like to stress the value of the introduced approach based on the complexity of problem. For the complex DIP to be solved using the dynamic support for ODIPs in one model, would give a state space of approximately $3.0 \cdot 10^{12}$. Clearly, this is not what the current computing power of a computer can handle in terms of memory also. At the same time, the largest decomposed problem has a state space of 3125 states and solves in less than 3 seconds. Moreover, any of the subprocesses that is optimized at the same level in the hierarchy can be solved in parallel, saving more time. Considering also the performance of the found policy for the small cases, we think the decomposition approach is a strong method that should be considered to optimize complex DIPs.

3.8. Related Work

Multiple authors have already established a ground layer of papers that point at the need for a method to solve and optimize decisions during a business process [Voorberg et al., 2021, Hasić et al., 2018, Batoulis et al., 2016]. Hasić et al. [2018] introduce five core principles for augmenting process and decisions, where one of these principles establishes the decision logic must act as a service to allow for enough scalability. The problem with all these papers is the scalability for an approach where decision and process are really incorporated. The suggested solutions have many sources of uncertainty making the problems very complex and this problem is often not addressed correctly. This paper adresses the need for a solution that is scalable using the decomposition of the problem.

The idea of decomposing a problem into smaller problems that together form a hierarchy and can solve a problem is not new. It has been introduced under the term of divide and conquer [Simon, 1981], which is also used a lot in the field of process mining [Nguyen et al., 2019]. But some also use the idea of Hierarchical Planning [Alford et al., 2016], or the Analytic Hierarchy process [Saaty, 1990]. Even in the field of BPM, this approach has been applied already multiple times [González-Ferrer et al., 2013, Kallestrup et al., 2014, Turetken et al., 2020]. As was already mentioned also in Chapter 2, this sort of research is mainly meant for the comprehensibility of processes in contrast with the optimization objective in this thesis. Hasić et al. [2018] also already introduced a list of principles that can be used to combine decisions and processes of which we follow four principles, except the decision logic exclusion principle that was already mentioned in the previous paragraph. For the application of DMN in this chapter, we also stress the introduction of uncertain information attributes. The basic assumption of DMN that attributes are always available to make a decision [Object Management Group, 2020], is not valid for this chapter. This new way of modeling adds a lot of opportunities to apply DMN. To use the way of structuring a process based on human experience was also already introduced [Claes et al., 2016]. The added value of the approach in this chapter is that we implement the idea in existing modeling approaches of the BPM field. Therefore, one can use these existing models and still use introduced optimization solutions such as the ODIP approach of Chapter 2.

The need to solve ever larger getting problems is the source of existence for the field of machine learning. Also in this field, it is clear that using decomposition methods based on hierarchy are very applicable. Hauskrecht et al. [2013] already acknowledge that one can decompose an MDP in a higher macro MDP responsible for main decisions, flowing back to low level MDPs. The field of hierarchical reinforcement learning promises to be able to learn how to break down a problem into smaller size problems [Merel et al., 2019, Botvinick, 2012]. This would be an interesting research direction to continue on and to compare with the approach of this chapter.

3.9. Discussion

The resulting approach of this chapter does not lead to optimality. In the case of optimality, all information should be considered in one large model. However, for complex DIPs this sort of models becomes intractable and hence the approach is a good solution for DIPs for which it would otherwise be impossible to optimize them. This approach is not necessarily the only way. For example, the process can also be optimized using a more complex optimization method such as deep reinforcement learning (vid. Chapter 5). However, the natural presence of hierarchy that often appears in complex DIPs is used perfectly in the approach of this chapter. Moreover, we believe that knowledge workers will more quickly adapt an approach where they can see what the target is for their specific department compared to using a DRL solution that returns a incomprehensible policy.

The reason that we combine CMMN with DRG notation is because of the crucial connection between decision making and information in DIPs. When trying to optimize the process of acquiring information it is rather crucial to know what is the goal that we are trying to optimize. DRG notation helps in fixing this goal and subgoals. Finally, by adding the uncertain information items in the DRG notation, we are capable of modeling the improvement of the information position during the process. The combination of DMN and CMMN is mentioned in literature but not often applied [Suchenia et al., 2019, Wiemuth et al., 2017]. On the contrary, there is another stream of literature that proposes the combination of BPMN and DMN [De Leoni et al., 2021, Janssens et al., 2016]. However, the use of DMN is much more focused on direct decisions that can constantly change again and depend on direct actions.

In this chapter we assume that the information structures can be built per subdecision based on targets of subgroups or departments in companies. We do think, however, that there are possibilities to extend the approach to a solution where one main information structure is also decomposed into smaller information structures. The problem here is that with one information structure it is not straightforward to define how crucial each of the subdecisions is to arrive at a correct final decision for the main decision.

3.10. Conclusion

In this chapter we have introduced an approach to optimize a complex DIP. A complex DIP is of such large size that there is too much information impacting the main decision for it to be computationally tractable. The approach decomposes the complex DIP in a set of smaller size ODIPs. These ODIPs are solved according to the optimization approach of Chapter 2. The approach allows to find a policy by starting at the lowest rank and working up through the hierarchy of processes. However, due to the complexity of complex DIPs it is not possible to find a optimal solution since the state space of such problems becomes intractable for the optimization method. Therefore, we believe that this is a valuable solution to a set of DIPs with which many companies are faced. The approach is built based on a combination of a CMMN model and a DRG model. The CMMN model models the workflow of the process where information is acquired. The DRG model defines the tree of subdecisions and their dependencies, such that the complex DIP can be decomposed.

A small example case has been introduced to show the effectiveness of the method compared to the optimal solution for that case. Moreover, a full scale complex DIP has been introduced for which a policy is calculated based on a set of ODIPS. For a small part of this complex DIP we also were able to compare the found policy to the full scale optimized policy. For both cases we derive the set of ODIPs that are optimized. We show in a comparison between the policies how well the approach is able to find a similar looking policy compared to the optimal policy. Besides, the profit in time is vast. The decomposed ODIPs are solved in a matter of seconds while even for the small example case the optimal solution takes long to solve. Finally, we see that the difference in expected profit between the optimal solution and the decomposed approach is small for this example case.

Appendix

3.A. Decision tree of weather attribute

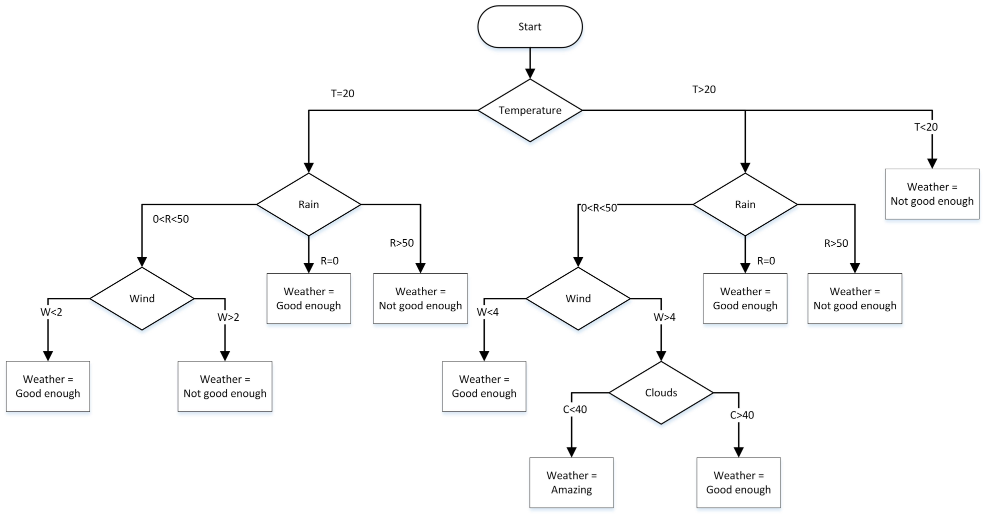


Figure 3.12

3.B. Complex DIP description

Any italic phrase in the following complex DIP refers to a task and any term in bold can be recognized as an information attribute.

Custom Bikes

A small company manufactures customized bicycles. Whenever the sales department receives an order, the order handling process starts. Due to the design complexity, every request to manufacture a customized bike denotes a separate order.

In a first phase, the order is registered and added to the file of the existing customer. If a new customer places an order, a customer file will be created and the order is added. Then, the order can be evaluated through (1) an engineering check, (2) profit check and (3) a financial check.

For the *engineering check*, based on the number of parts and the way to procure them, the engineers can determine the order type. The bike is divided into four subsections, each consisting of a certain number of parts: **Gears, Frame, Brakes, Steer**. Furthermore, any **custom parts** determine what type of bike it is. If the complete bike consists of less than 100 parts, either bought or custom made, the order is categorized as easy; If the bike has between 100 and 500 parts, the order is considered to be (1) easy: if all parts can be bought, (2) difficult: if some parts need to be custom built. If the order consists of more than 500 parts, it is considered complex.

Second, the *profit check* process has the following input attributes: **Number of custom parts**, the **customer history**, the **offered price** and the outcome of the **engineering check**. The estimated costs are equal to $2000 for each class higher in the engineering check, and $200 added up for each custom part on the bike. The estimated revenue is equal to $4000, $8000, $12000 or $16000 with a multitude of 4% discount depending on the customer's history. The expected profit on the bike is the revenue minus the cost.

Third, the *financial check* depends on the outcome of the **engineering check** and the **profit check** of the order. If the order is easy and the calculated profit of the order surpasses $1000, the order receives a financial OK status. A difficult order is only accepted if the profit amounts to at least $5000; if the profit ranges between $1000 and $5000, the order is attributed a Medium status. A complex order will only be accepted if the estimated profit exceeds $7000, and receives a Medium evaluation if the profit is between $5000 and $7000.

After the engineering and financial checks, a *senior manager* verifies the order to check whether it fits the style of the company. Orders with an OK **financial status** will be accepted, regardless of the manager's taste. Orders with a Medium financial evaluation will only be accepted if the **manager evaluation** is high. If the financial evaluation was not ok, the order will be rejected, unless it denotes a special purpose bike, which will always be produced. A special bike contains at least 2 **custom parts**.

In case the order is rejected, the customer receives a notification. If the order is accepted, the storehouse and the engineering department are informed.

The *storage department* immediately processes the part list of the order and checks the required quantity of each part. If the part is available in-house (inventory), it is reserved. Any number of **missing parts** is back-ordered. In that case, the lead engineer will first draft each part's requirements. We assume a **custom part** always has to be ordered. The part requirements are then submitted to three different manufacturers.

The quotes received from the manufacturers are then evaluated. This evaluation assesses each manufacturer based on: **payment terms, quality assurance, manufacturer reliability** and **delivery options**. For each manufacturer, a *company profile (A,B,C)* is determined that impacts the quotation procedure. These **company profiles** are evaluated together with the **company capability**, the *best manufacturer* is chosen for the whole set of back-order parts and the parts are ordered. If no manufacturer can be found for the set of parts, a back order failed error is thrown. At that moment, the company will contact the customer to inform him of the difficulties with the order and proposes him to submit a new order; the current order is canceled.

The **engineering department** starts with including the order in the production schedule and performs a number of assembly preparation tasks. Depending on this, the bike can either be produced in time or with a certain delay. The *final decision* of the process is whether to start the production depending on the **engineering department**, **manager decision** and **storage department**. If the order is canceled, the engineering department stops the preparation of the assembly, and the manufacturing planning is changed. When all parts are ready and the engineering department has planned the production of the bike, the bike is assembled either in time or with a delay. After assembly, the bike is shipped and invoiced.

3.C. Attribute distributions

Table 3.7: Probability distribution of attribute values for the attributes in the example case.

Attribute	*Possible Outcomes*							
	Value	*Prob*	*Value*	*Prob*	*Value*	*Prob*	*Value*	*Prob*
Marc	Yes	0.16	Maybe	0.48	No	0.36		
Joey	Yes	0.09	Maybe	0.42	No	0.49		
Anne	Yes	0.25	Maybe	0.50	No	0.25		
% Rain	0	0.25	30	0.25	60	0.25	90	0.25
Temperature	15	0.51	20	0.38	25	0.1	30	0.01
Wind speed	0	0.34	2	0.44	4	0.19	6	0.03
% Cloudiness	100	0.125	50	0.375	25	0.375	12.5	0.125

3.D. Case description Diagrams

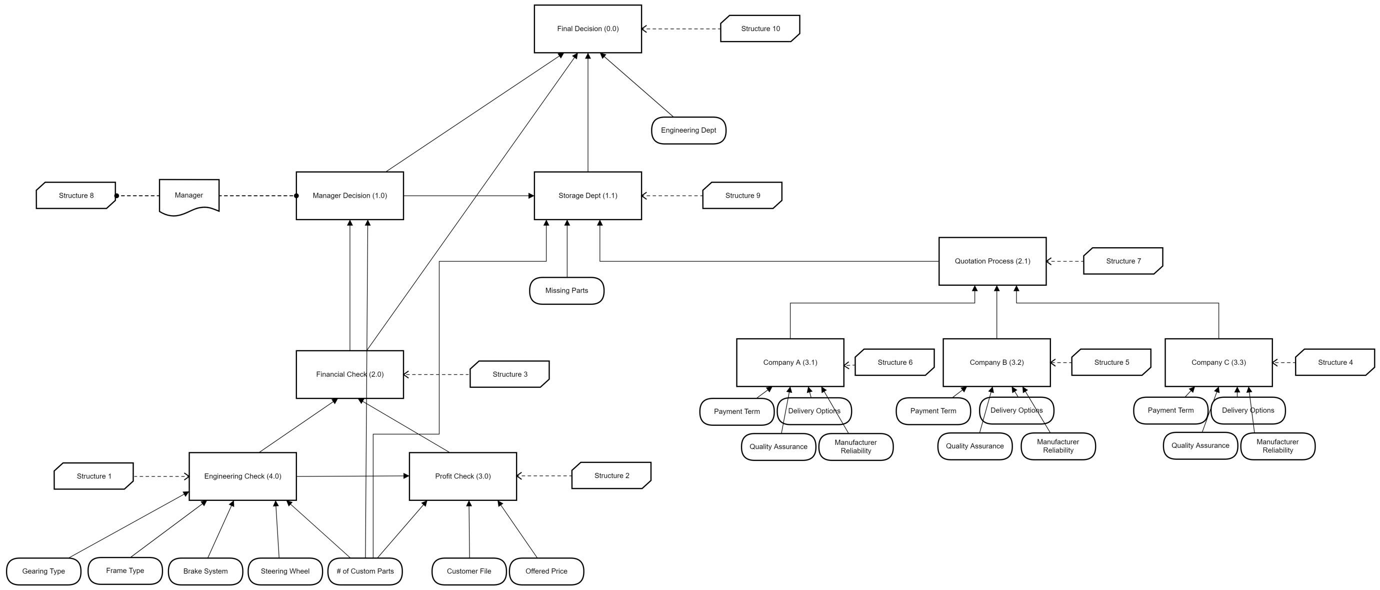

Final Decision (0.0)
Structure 10
Engineering Dept
Structure 8
Manager
Manager Decision (1.0)
Storage Dept (1.1)
Structure 9
Quotation Process (2.1)
Structure 7
Missing Parts
Company A (3.1)
Structure 6
Company B (3.2)
Structure 5
Company C (3.3)
Structure 4
Financial Check (2.0)
Structure 3
Payment Term
Delivery Options
Quality Assurance
Manufacturer Reliability
Payment Term
Delivery Options
Quality Assurance
Manufacturer Reliability
Payment Term
Delivery Options
Quality Assurance
Manufacturer Reliability
Structure 1
Engineering Check (4.0)
Profit Check (3.0)
Structure 2
Gearing Type
Frame Type
Brake System
Steering Wheel
# of Custom Parts
Customer File
Offered Price

Figure 3.13

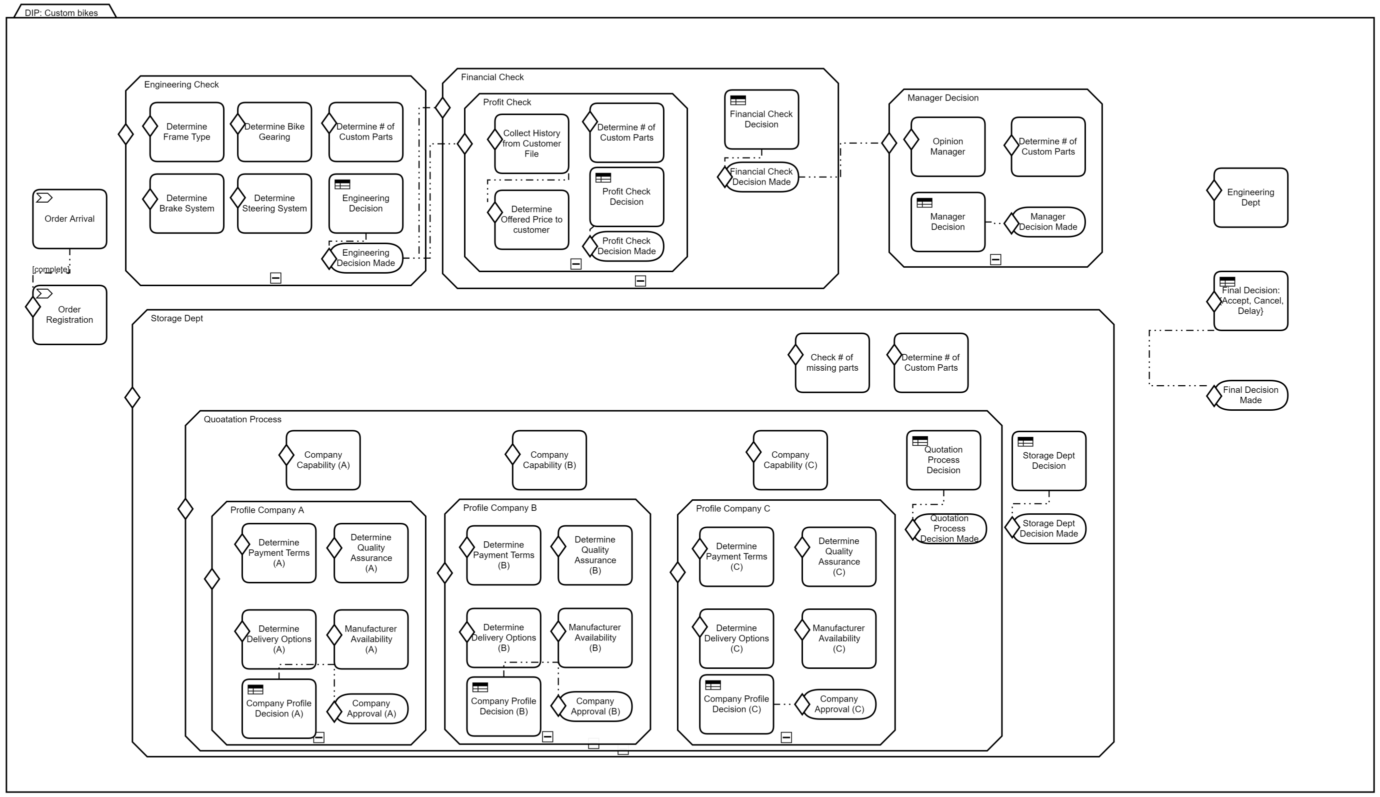

DIP: Custom bikes
Order Arrival
[complete]
Order Registration
Engineering Check
Determine Frame Type
Determine Bike Gearing
Determine # of Custom Parts
Determine Brake System
Determine Steering System
Engineering Decision
Engineering Decision Made
Financial Check
Profit Check
Collect History from Customer File
Determine # of Custom Parts
Determine Offered Price to customer
Profit Check Decision
Profit Check Decision Made
Financial Check Decision
Financial Check Decision Made
Manager Decision
Opinion Manager
Determine # of Custom Parts
Manager Decision
Manager Decision Made
Engineering Dept
Final Decision: {Accept, Cancel, Delay}
Final Decision Made
Storage Dept
Check # of missing parts
Determine # of Custom Parts
Quoatation Process
Company Capability (A)
Company Capability (B)
Company Capability (C)
Quotation Process Decision
Quotation Process Decision Made
Storage Dept Decision
Storage Dept Decision Made
Profile Company A
Determine Payment Terms (A)
Determine Quality Assurance (A)
Determine Delivery Options (A)
Manufacturer Availability (A)
Company Profile Decision (A)
Company Approval (A)
Profile Company B
Determine Payment Terms (B)
Determine Quality Assurance (B)
Determine Delivery Options (B)
Manufacturer Availability (B)
Company Profile Decision (B)
Company Approval (B)
Profile Company C
Determine Payment Terms (C)
Determine Quality Assurance (C)
Determine Delivery Options (C)
Manufacturer Availability (C)
Company Profile Decision (C)
Company Approval (C)

Figure 3.14
]

3.E. Case results

Table 3.8: Subprocess decomposition and acquisition costs per task/stage/subprocess for the main case (part A).

Process	*Task/Stage/Subprocess*	*Attribute*	*Costs*
Engineering Check (4.0)	Determine # of custom parts	**# of Custom Parts**	*4*
	Determine gearing type	**Gearing Type**	*2*
	Determine frame type	**Frame Type**	*2*
	Determine brake system	**Brake System**	*2*
	Determine steering system	**Steering system**	*2*
Profit Check (3.0)	Determine # of custom parts	**# of Custom Parts**	*4*
	Collect History from Customer File	**Customer File**	*2*
	Determine Offered Price to customer	**Offered Price**	*10*
	Engineering Check	$X_{(4.0)}$	*7.1*
Financial Check (2.0)	Engineering Check	$X_{(4.0)}$	*7.1*
	Profit Check	$X_{(3.0)}$	*7.5*
Manager Decision (1.0)	Financial Check	$X_{(2.0)}$	*7.7*
	Determine # of custom parts	**# of Custom Parts**	*4*
	Ask Manager	**Manager Opinion**	*5*

Table 3.9: Subprocess decomposition and acquisition costs per task/stage/subprocess for the main case (part B).

Process	*Task/Stage/Subprocess*	*Attribute*	*Costs*
Company Profile A (3.1)	Ask Payment Term (A)	**Payment Term A**	*9*
	Ask Quality Assurance (A)	**Quality Assurance A**	*7*
	Ask Delivery Options (A)	**Delivery Options A**	*5*
	Check Manufacturer Reliability (A)	**Manufacturer Reliability A**	*15*
Company Profile B (3.2)	Ask Payment Term (B)	**Payment Term B**	*9*
	Ask Quality Assurance (B)	**Quality Assurance B**	*7*
	Ask Delivery Options (B)	**Delivery Options B**	*5*
	Check Manufacturer Reliability (B)	**Manufacturer Reliability B**	*15*
Company Profile C (3.3)	Ask Payment Term (C)	**Payment Term C**	*9*
	Ask Quality Assurance (C)	**Quality Assurance C**	*7*
	Ask Delivery Options (C)	**Delivery Options C**	*5*
	Check Manufacturer Reliability (C)	**Manufacturer Reliability C**	*15*
Quotation Process (2.1)	Company Capability A	**Company A**	*8*
	Company Profile A	$X_{(3.1)}$	*26.8*
	Company Capability B	**Company B**	*8*
	Company Profile B	$X_{(3.2)}$	*23.9*
	Company Capability C	**Company C**	*8*
	Company Profile C	$X_{(3.3)}$	*21.6*
Storage Department (1.1)	Check # of missing parts	**Missing Parts**	*20*
	Determine # of custom parts	**# of Custom Parts**	*4*
	Quotation Process	$X_{(2.1)}$	*34.64*
Final Decision (0.0)	Manager Decision	$X_{(1.0)}$	*7.9*
	Storage Dept	$X_{(1.1)}$	*38.3*
	Check Engineering dept.	**Engineering Dept**	*3*

Table 3.10: Probability distribution of attribute values for the attributes in the main case.

Attribute	*Value*	*Prob*	*Value*	*Prob*	*Value*	*Prob*	*Value*	*Prob*
# of Custom Parts	*0*	*0.25*	*2*	*0.25*	*4*	*0.25*	*6*	*0.25*
Gearing Type	*20*	*0.25*	*50*	*0.25*	*160*	*0.25*	*200*	*0.25*
Frame Type	*30*	*0.25*	*45*	*0.25*	*60*	*0.25*	*70*	*0.25*
Brake System	*40*	*0.25*	*50*	*0.25*	*120*	*0.25*	*180*	*0.25*
Steering system	*5*	*0.25*	*10*	*0.25*	*15*	*0.25*	*100*	*0.25*
Customer File	*New*	*0.33*	*Existing*	*0.33*	*Frequent*	*0.33*		
Offered Price	*5000*	*0.25*	*10000*	*0.25*	*15000*	*0.25*	*20000*	*0.25*
Manager Opinion	*High*	*0.5*	*Low*	*0.5*				
Payment Term (A)	*10*	*0.25*	*20*	*0.25*	*30*	*0.25*	*40*	*0.25*
Quality Assurance (A)	*90%*	*0.33*	*95%*	*0.33*	*99%*	*0.33*		
Delivery Options (A)	*Same day*	*0.33*	*Next day*	*0.33*	*One week*	*0.33*		
Manufacturer Reliability (A)	*80%*	*0.33*	*90%*	*0.33*	*100%*	*0.33*		
Payment Term (B)	*10*	*0.4*	*20*	*0.2*	*30*	*0.2*	*40*	*0.2*
Quality Assurance (B)	*90%*	*0.2*	*95%*	*0.33*	*99%*	*0.47*		
Delivery Options (B)	*One week*	*0.3*	*Next day*	*0.7*	*Same Day*	*0*		
Manufacturer Reliability (B)	*80%*	*0.1*	*90%*	*0.4*	*100%*	*0.5*		
Payment Term (C)	*10*	*0.02*	*20*	*0.1*	*30*	*0.38*	*40*	*0.5*
Quality Assurance (C)	*90%*	*0.6*	*95%*	*0.3*	*99%*	*0.1*		
Delivery Options (C)	*Same day*	*0*	*Next day*	*0.2*	*One week*	*0.8*		
Manufacturer Reliability (C)	*80%*	*0.7*	*90%*	*0.2*	*100%*	*0.1*		
Company A	*Capable*	*0.4*	*Not Capable*	*0.6*				
Company B	*Capable*	*0.3*	*Not Capable*	*0.7*				
Company C	*Capable*	*0.2*	*Not Capable*	*0.8*				
Missing Parts	*0*	*0.25*	*1*	*0.25*	*2*	*0.25*	*3*	*0.25*
Engineering Dept	*In Time*	*0.8*	*Delayed*	*0.15*	*Declined*	*0.05*		

Table 3.11: Result of the Decomposed (DEC) policy for all subprocesses in the main case.

Subprocess	***Attribute***		***Outcome***					***Cost***
Engineering Check	$X_{(4.0)}$	*Value*	Easy	Difficult	Complex			*7.1*
		Prob.	0.25	0.73	0.02			
Profit Check	$X_{(3.0)}$	*Value*	1000	3000	5000	7000	9000	*7.5*
		Prob.	0.33	0.24	0.10	0.24	0.09	
Financial Check	$X_{(2.0)}$	*Value*	OK	Medium	Not OK			*7.7*
		Prob.	0.21	0.21	0.58			
Manager Decision	$X_{(1.0)}$	*Value*	Accept	Reject				*7.9*
		Prob.	0.58	0.42				
Company Profile A	$X_{(3.1)}$	*Value*	Excellent	Good	Bad			*26.8*
		Prob.	0.05	0.44	0.51			
Company Profile B	$X_{(3.2)}$	*Value*	Excellent	Good	Bad			*23.9*
		Prob.	0.03	0.49	0.48			
Company Profile C	$X_{(3.3)}$	*Value*	Excellent	Good	Bad			*21.6*
		Prob.	0	0.67	0.33			
Quotation Process	$X_{(2.1)}$	*Value*	3	2	1	0		*34.64*
		Prob.	0	0.02	0.43	0.55		
Storage Dept	$X_{(1.1)}$	*Value*	In Time	Delayed	Declined			*38.3*
		Prob.	0.29	0.16	0.55			
Final Decision	$X_{(0.0)}$	*Value*	Accepted	Delayed	Declined			*27.47*
		Prob.	0.41	0.16	0.42			

Table 3.12: Fraction of cases for which the Decomposed policy performs the specific tasks and acquires the information.

Subprocess	***Task/Stage/Subprocess***	Fraction
Engineering Check	Determine # of custom parts	1
	Determine gearing type	0.074
	Determine frame type	0.0625
	Determine brake system	0.4375
	Determine steering system	1
Profit Check	Determine # of custom parts	1
	Collect History from Customer File	1
	Determine Offered Price to customer	0.08
	Engineering Check	0.67
Financial Check	Engineering Check	0.24
	Profit Check	1
Manager Decision	Financial Check	1
	Determine # of custom parts	1
	Ask Manager	0.18
Company Profile A	Ask Payment Term (A)	1
	Ask Quality Assurance (A)	0.7525
	Ask Delivery Options (A)	1
	Check Manufacturer Reliability (A)	0.5
Company Profile B	Ask Payment Term (B)	1
	Ask Quality Assurance (B)	1
	Ask Delivery Options (B)	0.59
	Check Manufacturer Reliability (B)	0.33
Company Profile C	Ask Payment Term (C)	1
	Ask Quality Assurance (C)	0.58
	Ask Delivery Options (C)	0.676
	Check Manufacturer Reliability (C)	0.343
Quotation Process	Company Capability A	0.8
	Company Profile A	0.32
	Company Capability B	0.8
	Company Profile B	0.24
	Company Capability C	1
	Company Profile C	0
Storage Dept	Check # of missing parts	0.215
	Determine # of custom parts	1
	Quotation Process	1
Final Decision	Manager Decision	1
	Storage Dept	0.49
	Check Engineering dept.	0.58

4

Information Acquisition for Service Contract Quotations by Repair Shops

4.1. Introduction

As for many capital goods, an aircraft is *modularly* designed to ensure that it can be maintained quickly and cost-effectively. Aircraft maintenance mainly consists of the replacement of certain components by ready-for-use components that are certified as being in good condition, while the removed components are maintained separately by specialized *repair shops*.

The market for non-military component maintenance, repair and overhaul (CMRO) generates an annual turnover above \$9 billion (Aviation week, 2011). In this market, a large fraction of repairs is conducted under long-term service agreements. Commercial airlines outsource some 70% of their component maintenance [Carpenter and Henderson, 2008]. In particular, after forecasting their need for repair services for each component for the coming period (e.g., two years), airliners share these needs with several CMRO providers by sending out a *request for quotation* (RFQ). Each RFQ contains information regarding the type of component to be repaired, the duration of the agreement, as well as some other information relevant to the services to be delivered. The providers then place quotations, i.e., they indicate at

This chapter is based on Voorberg et al. [2023].

what price they would be willing to deliver the services. The airliner then selects the most favorable quotation and enters an agreement with the corresponding provider to carry out the services.

It is crucial for providers to respond to the right RFQs with appropriately priced quotations. Airliners share RFQs with a range of providers in order to secure the most favorable price. However, arriving at a proper quotation involves collecting pieces of information from various sources, and collecting each piece of data is time-consuming. Therefore, it is typically not cost-effective to collect all information for each RFQ.

In this chapter we gain insights into how CMRO providers should collect information for RFQs. In order to model the problem, we propose a model for *dynamic information acquisition for profit maximization* (DIA model), drawing upon literature on optimal information acquisition [e.g., Mookerjee and Dos Santos, 1993, Dos Santos and Mookerjee, 1993]. This model can be seen as the Operations view on a DIP. DIPs are defined in the field of BPM to model the process, information and decisions and their interaction. The DIA model focuses on the optimization of such interactions.

By further refining the DIA model, we propose a model for quotation optimization (QO model). The QO model and the case study are based on the quotation process that has been observed at an existing independent aviation repair shop, which we refer to as Fokker Services. The QO model considers the collection of various pieces of information that Quotation Agents (QAs) may use before offering a quotation. The presented model reflects the *key challenges* of the real-life quotation process:

- Quoting a high price increases profitability if the contract is won, while reducing the chances of winning the contract.
- To arrive at very precise estimates of optimal quotation prices, many pieces of information that are not directly accessible are needed, but due to the high number of RFQs there is not enough time to collect all this information for every quotation. Hence, a trade-off arises between making a good decision and making an efficient decision.
- Pieces of information are collected one by one, and in view of point 1 the process should be terminated when it becomes unlikely that a profitable and winning quotation can be made.

The QO model utilizes a *function* for valuing quotation prices that is based on company interviews at Fokker Services. This function also reflects challenge 1. Each attribute that can be collected is associated with a variable of this value function. When the quotation process for a newly arrived RFQ is started, the value of these variables is unknown, though estimated probability distributions are available based on earlier similar cases. Collecting an attribute is then modelled as retrieving the actual value of the associated variable, in line with literature on optimal information acquisition [e.g., Mookerjee and Dos Santos, 1993, Dos Santos and Mookerjee, 1993]. Our model features retrieval costs for each information attribute. These costs are proportional to the effort required to collect the information, which ensures that optimal solutions to the model make efficient use of time, reflecting challenges 2 and 3.

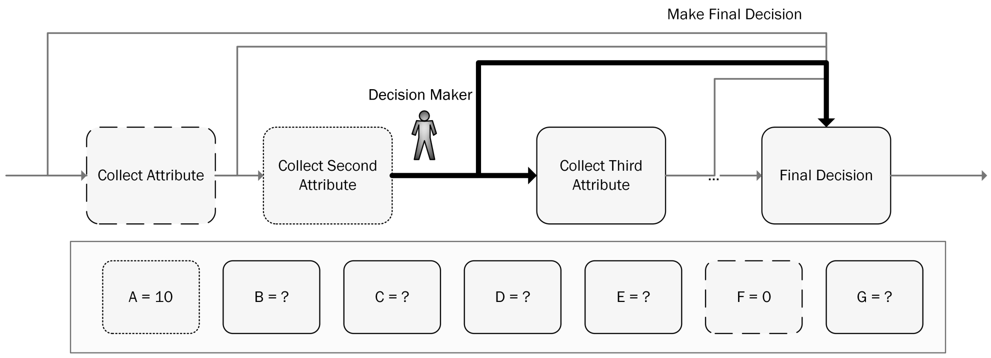


Figure 4.1: Overview of the Decision-making process: The decision maker has collected attributes F and A respectively and has to choose now between making a final decision or collecting a third attribute.

When an RFQ for a specific component type comes in, the model is used to guide the human decision maker. At each step the decision maker either retrieves an additional piece of information, or finalizes the process (by quoting a price, or by deciding not to quote). In Figure 4.1 we give a schematic overview of the general procedure. For guiding the decision maker, we consider various policies with respect to the order and the number of attributes to be retrieved. For all policies, the final decision is dependent on the value of the attributes that are retrieved. But the policies differ with respect to their advancedness and comprehensibility. The most advanced policy is as follows:

- Policy 1 - Fully dynamic policy: This policy utilizes stochastic dynamic programming to solve instances of the model to mathematical optimality. Optimal solutions recover attributes completely dynamically: the choice of which attribute to retrieve next depends on the values of all previously retrieved attributes.

This policy 1 is also used in the Optimizable DIP approach in Chapter 2. In that chapter, the focus was on how to translate DIP architecture from the field of BPM into an MDP that is solved using policy 1. This chapter focuses more on the value of the solution in the environment of quotation optmization.

The fully dynamic policy gets the most value out of the attributes it retrieves. However, from the perspective of a human that repeatedly uses the model to arrive at quotations for a component type, the dynamic policy has a disadvantage: The order in which the attributes are retrieved is typically different each time that a quotation is prepared. When working with the approach, this may hamper the decision maker to build a *routine*. To address this we also develop policies that cater more to the preferences of humans when carrying out the process:

- Policy 2 - Fixed-order policy: Under this policy, attributes are always retrieved in the same order. After each attribute is retrieved, we decide optimally on whether it is beneficial to retrieve the next attribute, or whether we should make a final quotation.
- Policy 3 - Fixed-attributes policy: Under this policy, the decision maker always retrieves the same subset of attributes and only optimizes the final quotation decision.

Policy 1 makes an optimal trade-off between the effort invested, expressed in costs, and the quality of the final quotation, expressed in a reward. An important finding of our work, however, is that the trade-off accomplished with Policy 2 is almost as beneficial as under policy 1: the quality of Policy 2 is more than 96% of that of Policy 1 for the QO model. This is measured in terms of expected profit improvement on top of a naive policy that always collects all information before making the optimal final decision. For Policy 3, however, the solution quality suffers substantially when keeping the average effort investment fixed: The profit improvement is about 61% of Policy 1. Our main insight is thus that it is important to make final decisions based on partial information, but it is not crucial to make the order in which attributes are retrieved dependent on the value of retrieved attributes.

The main contributions of this chapter are as follows:

- We formulate a dynamic information acquisition model for profit maximization (DIA model).
- We introduce the quotation process of a repair shop as a specific implementation to study the DIA model (QO model).
- We study the effect of fixing the order (Policy 2) or the complete set (Policy 3) of acquiring information attributes to simplify the process for a human decision maker compared to the optimal solution (Policy 1).

The remainder of this chapter is organized as follows. Section 4.2 will introduce the current literature in this area of research. In Section 4.3, we will introduce the DIA model that is used to solve our decision making problem and present the dynamic program that can solve this model. Section 4.4 introduces the QO model that is a detailed implementation of the DIA model. In Section 4.5 we numerically study instances of this QO model for the three different policies that were introduced. Section 4.6 will conclude the work.

4.2. Related Work

We propose and study a quantitative model of the quotation process for repair service providers. Our contribution lies primarily in studying this process. We propose a quantitative model of the process and investigate the relative merits of various policies. To place this model in a broader perspective, we provide a detailed review of work that develops and solves quantitative models for decision making by service providers.

Quantitative models for decision making by service providers: Research on decision making in repair shops, and more generally on the component repair process, is well-developed, but we are not aware of prior work that studies the quotation process for service providers or other related environments. The paper by Deprez et al. [2020] is the first paper focusing on using predictive information to set a price for maintenance service contracts. They focus on the prediction of failure behaviour of components and how to take this into account when building a maintenance portfolio. This requires specific data of the components. In the aircraft sector, but also others sectors, this data is hard to collect for the service provider. Hence, our model focuses more on the quotation procedure of a given service provider and the competition with other service providers.

In the past years Driessen et al. [2016] and Hu et al. [2018] have recognised the need in maintenance optimization to look at the bigger picture of using capital goods. The use of capital goods does not only involve maintenance costs and can also be organised in different ways through, for example, *service level agreements* (SLA) with service providers. Papers such as Van der Auweraer and Boute [2019] already show the value of combining steps in maintenance optimization through forecasting and maintenance policies.

In line with observations in Driessen et al. [2016], key decisions for repair shops pertain to inventory control of shop-replaceable units and capacity dimensioning in repair shops. Below we discuss related work in these two categories, and we then place the problem studied in this chapter in a broader context of literature.

Inventory control for repairable components themselves is a well-researched topic (see Basten and van Houtum [2014] for an overview), but it is a problem that is not in the scope of commercial repair shops since they typically do not own components themselves, but rather repair components owned by operators. The inventory control of shop-replaceable units (SRUs) is however a key decision made by repair shops [van Jaarsveld et al., 2015], and this decision is studied in multi-indenture inventory models. Muckstadt [1973] was the first to study this problem. For a detailed overview of this stream, we refer to Topan et al. [2020], Bülbül et al. [2019] and de Jonge and Scarf [2020].

The capacity of a repair shop is mainly determined by the number of skills and maintenance engineers, but in some cases specialized equipment can also be a limiting factor. Quantitative models have been developed to support the decision on appropriate long term capacities in combination with inventory control [Sleptchenko et al., 2003] and to aid with the related cross-training decision [Sleptchenko et al., 2019]. Once capacity is available, it can be allocated to certain tasks (e.g., component types).

Next to a review on quantitative models for decision making by service providers, we also want to position our research in terms of methodology. Repair shops receive thousands of RFQs per year and the goal of this chapter is developing and testing models that can aid in *time-efficiently* identifying the promising RFQs and to respond to those RFQs with appropriate quotations. The key mode employed by quotation officers to save time while preparing quotations is *not necessarily collecting every piece of information for every RFQ*, but instead in some cases to terminate information collection early and offer a quotation before all information is collected. Thus, from a *methodological* point of view, the model and algorithms proposed in this chapter are related both to the multi-attribute stopping problem and to the information acquisition problem, and we will review the key related works in these streams next.

Multi-attribute stopping problem: The problem of information acquisition for quotation optimization, as studied in this chapter, is related to the field of multi-attribute stopping problems. A multi-attribute stopping problem consists of a decision maker that is offered a list of alternative decisions sequentially and only once. As soon as one alternative is chosen, the problem has finished. For every alternative, the decision maker can pay to collect a number of attributes that help in estimating the outcome of this specific alternative. For every new alternative, these attributes can again be collected. The key property of a stopping problem is that alternatives are offered only once and sequentially, i.e., every decision alternative is (permanently) accepted or rejected.

For multi-attribute stopping problems with additive (separable) reward functions, Lim et al. [2006], Smith et al. [2007] have shown the optimality of a threshold policy, given a fixed order of attribute retrieval. Klabjan et al. [2014] derived analytic results on the optimal policy with symmetrically distributed attributes and linear additive reward functions. This research area acknowledges the lack of dynamic orders for attribute retrieval. In this chapter, we do allow for choosing dynamic orders (under Policy 1).

Information acquisition problem: In an information acquisition problem, a decision has to be taken from a set of alternatives. Attributes can be collected that affect the outcome of all decision alternatives. Furthermore, any decision alternative can be chosen at all times. This allows for acquiring information attributes in a dynamic way, such that one can work more efficiently to the optimal alternative as discussed in Section 4.1. Moreover, there is the constant trade-off between paying for more information or deciding on a final decision alternative already.

Explicit modelling of information acquisition dates back to Moore and Whinston [1986, 1987]. The approach proposed in these papers relies on creating a partitioning of remaining possible information states by collecting information such that each partition has only one optimal decision alternative and has been used in situations where a case must be assigned to the correct cluster [Hall et al., 1986].

The approach proposed by Mookerjee and Dos Santos [1993] and Dos Santos and Mookerjee [1993] is much closer related to methods developed in this chapter. For example, as in this chapter, Mookerjee and Dos Santos [1993] use random variables to model the available information on attributes *before* collecting case-specific information on the attribute, while retrieving the value of the attributes is modelled as revealing the actual value of that random variable. The focus in this stream of literature is mostly on decision trees/clustering [Mookerjee and Mannino, 1997b, Mookerjee and Dos Santos, 1993, Mookerjee and Mannino, 1997c]. The main methodology used in this chapter is stochastic dynamic programming, and this yields criteria for taking final decisions that are focused on optimization instead of classification.

In this chapter, we contribute with a new type of information acquisition problems in (to our knowledge) a new domain: The optimization of quotation processes. We discuss in detail the quotation process, how it can be modelled as an information acquisition problem, and how to estimate attribute values in a practical setting. Additionally, to model quotation optimization problems appropriately, we propose a simple *dynamic information acquisition for profit maximization* model. This DIA model draws substantially from seminal work on information acquisition [e.g., Mookerjee and Dos Santos, 1993], but it departs from prior literature because the objective of the final decision is to maximize expected profit. Prior literature focuses on correctly classifying a case which, while related, is not mathematically equivalent to profit maximization. Using this objective of profit maximization, we introduce three different policies to compare the complexity of policies with the increase in profit. In this way, we are able to decide what is the best policy to be used by human decision makers concerning both complexity and profitability.

4.3. Information Acquisition and Decision making

In this section, we formally introduce the DIA model (Section 4.3.1). We introduce all necessary variables and discuss their role. Section 4.3.2 contains the dynamic program that is formulated to solve the acquisition problem for small and discrete instances of the model. In Section 4.3.3 we introduce instances of the DIA model for which we prove an optimal order of information collection. Section 4.3.4 introduces a toy example to show how we solve the dynamic program for a specific instance.

4.3.1 Dynamic Information Acquisition model

Consider a decision maker who needs to optimize a decision for a specific *case* (e.g., determining an appropriate quotation in response to an RFQ). The goal is to select a single decision $f \in \mathcal{F}$. Suppose $\mathcal{F}$ contains finitely many alternatives. For every decision $f \in \mathcal{F}$, we define the reward function $r_f(\cdot)$. For every $f \in \mathcal{F}$, this reward function depends on various attributes, indexed by $i \in I$, where I is the set of attributes. At the start of the process, the value of these attributes is uncertain, and the precise values can only be learned by making costly time investments. Following literature [e.g. Dos Santos and Mookerjee, 1993], we model attributes as random variables: learning the attribute value corresponds to revealing the actual value of the random variable. The decision maker seeks to maximize expected *profit*, which is the expected reward resulting from the final decision net the costs incurred for retrieving attributes, and hence needs to weigh the expected additional reward that results from acquiring more information and the costs incurred for retrieving that information.

The decision maker has access to the probability distribution for each attribute; the corresponding random variable is denoted by X_i. All X_i are mutually independent. (This probability distribution may for example be constructed by analysing previous, similar *cases* treated earlier by the company.)

The expected reward $r_f(\cdot)$ is a function of the attributes. To make this explicit, let $R_i := R(X_i)$ be the support of X_i, i.e., the set of values that can be taken on by attribute $i \in I$, and let $S = R_1 \times \ldots \times R_{|I|}$ be the Cartesian product of support sets. Then for every $f \in \mathcal{F}$, $r_f : S \to \mathbb{R}$. Furthermore, with slight abuse of notation, we will denote by $r_f(X_1, \ldots, X_{|I|})$ the reward of deciding $f \in \mathcal{F}$. (The reward

$r_f(X_1, \ldots, X_{|I|})$ depends on $X_i, i \in I$, hence it is a random variable.)

The decision maker can retrieve attributes in any sequence. It takes the decision maker τ_i time units to retrieve attribute $i \in I$. Costs per unit of time are denoted by γ. At any point, the decision maker can decide to make a final decision, $f \in \mathcal{F}$, that ends the process.

For convenience we denote by $s_i = \perp$ the situation where attribute $i \in I$ has not yet been learned. With this notation in hand, the knowledge about attribute $i \in I$ can be represented by $s_i \in \tilde{R}_i$, with $\tilde{R}_i = R_i \cup \{\perp\}$. Moreover, define the knowledge state of the decision maker by $s = \{s_i\}_{i \in I}$, with $s_i \in \tilde{R}_i$. For any s, define $I(s)$ as the set of attributes with unknown values, i.e., $I(s) = \{i \in I | s_i = \perp\}$.

The decision maker can make the final decision $f \in \mathcal{F}$ before all information is acquired. In those cases, the expected reward is expressed as a conditional expectation. In particular, for any knowledge state s, this expected reward equals

$$\tilde{r}_f(s) := E[r_f(X_1, \ldots, X_{|I|}) | \forall i \notin I(s) : X_i = s_i] \tag{4.1}$$

(We condition on all variables that are not in $I(S)$, which are precisely the attributes with known values.)

4.3.2 Stochastic dynamic program

The Dynamic Information Acquisition problem can be solved using a stochastic dynamic program (SDP). In this section, we introduce recursive equations that may be used to find optimal decisions for this SDP when the random variables are discrete and have small supports: R_i is countable and small. Returning to the definition of a knowledge state $s = \{s_i\}_{i \in I}$, we define the state space $\tilde{S} = \tilde{R}_1 \times \tilde{R}_2 \times \ldots \times \tilde{R}_{|I|}$.

The set $I(s)$ contains precisely the attributes that can be retrieved in state s. For any $s \in \tilde{S}$, retrieving attribute $i \in I(s)$ results in the following transition probabilities:

$$P_i(s, s') = \begin{cases} P(X_i = s'_i) & \text{if } \forall j \in (I \setminus i) : s_j = s'_j \\ 0 & \text{otherwise} \end{cases},$$

with $s' \in \tilde{S}$. Revealing attribute i results in a state transition from s to s', where $s_j = s'_j$ for all attributes except attribute i.

Now, let $V(s)$ denote the expected optimal profit incurred until the end of the process when starting in state $s \in S$. $V(s)$ can be computed recursively as follows, using $\tilde{r}_f(s)$ as defined in (4.1):

$$V(s) = \max\{\max_{i \in I(s)} \tilde{V}(s,i), \max_{f \in \mathcal{F}} \tilde{r}_f(s)\}, \qquad s \in \tilde{S} \tag{4.2}$$

$$\tilde{V}(s,i) = -\gamma\tau_i + \sum_{s' \in \tilde{S}} P_i(s,s') \cdot V(s'), \qquad s \in \tilde{S}, i \in I(s) \tag{4.3}$$

For $s \in S$, (4.1) and (4.2) simplify as follows:

$$\tilde{r}_f(s) = r_f(s), \qquad s \in S, f \in \mathcal{F} \tag{4.4}$$

$$V(s) = \max_{f \in \mathcal{F}}\{r_f(s)\}, \qquad s \in S \tag{4.5}$$

Using (4.4) – (4.5) as a starting point, the optimal actions for each state $s \in \tilde{S}$ can be computed using backward recursion. In the following section we prove for a specific situation an optimal order of collecting information.

4.3.3 Analysis

In this section, we give two specific instances of the DIA model for which we can prove an optimal order of collecting information. These two instances make strong assumptions on the size of the problem and the random variables, X_i.

Let X_1 and X_2 be two independent normal random variables, $X_1, X_2 \sim N(0,\sigma)$. Furthermore, to reveal the sample value of those random variables we pay costs $c_i = \gamma\tau_i$, where $c_1 > c_2$. There are two decisions that terminate the decision making process: $\mathcal{F} = \{0,1\}$. A state is described by $s = (s_1, s_2)$ and the possible actions are $\mathcal{F} \cup I(s)$. The final reward function is $r_f(s_1,s_2) = (s_1 + s_2)f$. Based on this reward function, the final decision gives either no reward ($f = 0$) or $s_1 + s_2$ ($f = 1$). The value function for the starting state $(\perp,\perp)$ is as follows (cf. (4.2) – (4.3)):

$$V(\perp,\perp) = \max \begin{cases} \int_{s_2}\int_{s_1}(s_1+s_2)dF(s_1)dF(s_2) \\ \int_{s_1}(V(s_1,\perp) - c_1)dF(s_1) \\ \int_{s_2}(V(\perp,s_2) - c_2)dF(s_2) \\ 0 \end{cases}$$

We can prove the following result (the proof is given in Appendix 4.B).

Theorem 1. *For the above described instance of the DIA model there exists an optimal policy for which it holds that X_1 is never retrieved before X_2.*

Similarly, let X_1 and X_2 be two independent normal random variables, $X_1 \sim N(0, \sigma_1)$, $X_2 \sim N(0, \sigma_2)$, where $\sigma_2 \geq \sigma_1$. Furthermore, to reveal the sample value of those random variables we introduce costs $c_i = \gamma\tau_i$, where $c_1 = c_2 = c$. A state is described by $s = (s_1, s_2)$ and the possible actions are: $\mathcal{F} \cup I(s)$. The final reward function is $r_f(s_1, s_2) = (s_1 + s_2)f$. We then obtain the following result (the proof is given in Appendix 4.C).

Theorem 2. *For the above described instance of the DIA model there exists an optimal policy for which it holds that X_1 is never retrieved before X_2.*

Due to the specific properties of both random variables being independent and identically (normally) distributed and a special symmetric final reward function, we can derive the special properties for these instances of the DIA model. However, in general such properties cannot be proven.

4.3.4 Small example

In this section, we use backward recursion to compute optimal decisions for a toy example. The goal is moreover to illustrate that optimal information acquisition may be crucial for profitability in some cases.

Suppose there is a decision-making process where two information attributes, X_1, X_2, can be retrieved. Hence, $|I| = 2$. The outcome of both variables is boolean referring to a yes(1)/no(0) answer. Suppose $P(X_i = 0) = P(X_i = 1) = 0.5, \forall i \in I$. The final decisions that one can take is to discard or accept, $\mathcal{F} = \{0, 1\}$. The reward depends on the final decision and the outcome of the two attributes: $r_f(s_1, s_2) = 9(f \cdot (|s_1 - 2s_2| - s_1 s_2) - f)$, i.e., $r_f(0,0) = -9f$, $r_f(1,0) = 0$, $r_f(0,1) = 9f$, $r_f(1,1) = -9f$. This means that the reward can be 9, 0 or -9. The retrieval costs are $\gamma\tau_1 = 1, \gamma\tau_2 = 2$.

For backward recursion we start from the states $s \in S$ where we have all information, and we compute the optimal decisions from (4.4) and (4.5), see Table 4.1. A positive profit is attained only when deciding to accept, if the first attribute returns a *no* and the second attribute returns a *yes*.

Table 4.1: Value of states $s \in S$

$s = (s_1, s_2)$	$V(s) = \max_{f \in \mathcal{F}}\{r_f(s)\}$	Opt. action
$(1,0)$	$\max\{r_1(1,0), r_0(1,0)\} = \max\{0,0\} = 0$	$f = 0$
$(0,0)$	$\max\{r_1(0,0), r_0(0,0)\} = \max\{-9,0\} = 0$	$f = 0$
$(1,1)$	$\max\{r_1(1,1), r_0(1,1)\} = \max\{-9,0\} = 0$	$f = 0$
$(0,1)$	$\max\{r_1(0,1), r_0(0,1)\} = \max\{9,0\} = 9$	$f = 1$

Based on the optimal rewards of the final states in Table 4.1, we can use the recursive equations (4.2) and (4.3) of Section 4.3.2 to do backward recursion and derive the profit for states where one of the attributes is still uncertain. For example, in state $(\perp, 0)$ we must choose between retrieving attribute $i = 1$, or making a final decision:

$$\tilde{V}((\perp,0),i) = -\gamma\tau_i + 0.5 \cdot V(1,0) + 0.5 \cdot V(0,0) = -1 + 0 = -1$$
$$V(\perp,0) = \max\{\tilde{V}((\perp,0),i), \tilde{r}_1(\perp,0), \tilde{r}_0(\perp,0)\} = \max\{-1,-4.5,0\} = 0$$

Table 4.2 contains the remaining states where one of the attributes is still uncertain and shows what the optimal action and expected profit is for those states.

Table 4.2: Value of states with uncertainty

$s = (s_1, s_2)$	$V(s) = \max\{\max_{i \in I(s)} \tilde{V}(s,i), \max_{f \in \mathcal{F}} \tilde{r}_f(s)\}$	Opt. action
$(0,\perp)$	$\max\{\tilde{V}((0,\perp),2), \tilde{r}_1(0,\perp), \tilde{r}_0(0,\perp)\} = \max\{4.5-2, 0, 0\} = 2.5$	Retrieve X_2
$(1,\perp)$	$\max\{\tilde{V}((1,\perp),2), \tilde{r}_1(1,\perp), \tilde{r}_0(1,\perp)\} = \max\{0-2, -4.5, 0\} = 0$	$f = 0$
$(\perp,0)$	$\max\{\tilde{V}((\perp,0),1), \tilde{r}_1(\perp,0), \tilde{r}_0(\perp,0)\} = \max\{0-1, -4.5, 0\} = 0$	$f = 0$
$(\perp,1)$	$\max\{\tilde{V}((\perp,1),1), \tilde{r}_1(\perp,1), \tilde{r}_0(\perp,1)\} = \max\{4.5-1, 0, 0\} = 3.5$	Retrieve X_1

Knowing the optimal action and expected profit for these states with one uncertain attribute, we can now derive the optimal action from the knowledge state where everything is uncertain.

$$\begin{aligned} V(\perp,\perp) &= \max\{\tilde{V}((\perp,\perp),1), \tilde{V}((\perp,\perp),2), \tilde{r}_1(\perp,\perp), \tilde{r}_0(\perp,\perp)\} \\ &= \max\{0.25, -0.25, -2.25, 0\} = 0.25 \end{aligned}$$

Hence, we can conclude that without the possibility of information acquisition this process is not rewarding with an expected profit of -2.25 when accepting and 0 when always discarding. However, if one pays an additional fee for unveiling one or two of the attribute values, starting with attribute 1, it is possible to obtain an expected profit of 0.25. This case is exemplary for the value of information acquisition. Adding the option of retrieving the value of one of the attributes in the process (against a cost), actually increases the expected profit of this process to a level where it becomes profitable to act upon. Similarly, we will see that for Quotation Optimization problems profit increases when time is invested in retrieving the value of important attributes, while retrieving everything or just quoting upfront is not rewarding.

4.4. QO model

In accordance with the introduction, this section considers an existing case at a repair shop of Fokker Services that deals with aircraft component maintenance contracts. In this process, the quotation agent determines the price that will be offered to the (future) client for each single component repair as long as the contract lasts. The quotation agent faces a delicate trade-off: Quoting low prices may result in losses for the repair shop because repair costs cannot be covered. Quoting high prices may result in not winning the contract. There are several information sources that can aid in getting this trade-off right, but it takes too much time to consult all these sources for each and every RFQ, resulting in additional trade-offs. This section provides a detailed modeling of how the DIA model may be employed to support decision makers in these various trade-offs.

4.4.1 Scope

The QO model is a refined version of the DIA model. Recall that a model based on the DIA model is characterized by final decisions $\mathcal{F}$, a reward function $r_f(\cdot)$, and the attributes $i \in I$ with corresponding random variables X_i and retrieval time τ_i.

Note that repair shops of service providers are typically certified for a range of component types [cf. van Jaarsveld et al., 2015], e.g., landing gears, leading edges, servomotors, integrated driver-generators, coffee makers, etc. There is substantial heterogeneity between component types, while repairs of components of the same type are relatively homogeneous: For example, landing gear overhaul may take between $300 - 500$ hours and between \$10000 and \$30000 in parts, while a tail-wing servomotor overhaul may take between 30 and 65 hours, and \$2000 − \$7000\$ in piece parts, where piece parts are the one-piece parts that are part of a component.

Accordingly, we envision that service providers that seek to streamline the quotation process should develop a number of model instances from the QO model: one QO model instance per component type. Here, a component type corresponds to several closely related part numbers, e.g. a specific range of servomotors from a specific OEM. Each model instance is constructed while drawing from data collected on earlier RFQs and repairs for components of that type (i.e., those part numbers).

Each arriving RFQ is then treated as an instance of the corresponding QO model, depending on the type of the component for which the RFQ is requested. As repair shops typically collect a lot of data on repairs, especially on their key repairs, this approach may yield models that are sufficiently accurate and specific.

In the remainder of this section, we discuss a specific QO model. The model represents the real-world quotation process for a specific component type. While stylized, the model contains the key constituents that play a role in the real process of Fokker Services. Hence, one can interpret the QO model also as a use case of the DIA model.

4.4.2 Quotation process

A good quotation price trades off the benefit of winning the contract against a profitable price and the risk of not winning the contract. Some general information is typically available on the component type that the RFQ pertains to, e.g., in the head of an experienced QA or in a company database or spreadsheet.

On top of this general information, QAs typically collect additional RFQ-specific information from various sources. While this additional information is crucial in order to arrive at suitable quotation prices, there is typically not enough time to collect all information for each and every quotation.

The approach for distilling quotations for component-type and RFQ-specific information may be based upon a simple heuristic approach. For example, the QA might have some margin to the estimated repair costs; this margin might depend on the expected amount of competition. However, we will see that it is also possible to capture the trade-off between a low and a high quotation price in a relatively simple optimization model. Such optimization models are actually already employed by repair shops to arrive at more consistent quotation prices, e.g., as a simple spreadsheet model.

In either case, QAs may explain which information attributes are typically gathered before sending out a quotation, and how those attributes relate to the estimated profit of quoting a specific price.

4.4.3 Attributes

Based on discussions at a repair shop for aircraft components, we introduce seven RFQ-specific attributes that may be gathered to arrive at good quoted prices. For each attribute, we discuss how it moderates the estimated profit of quoting a specific price, which information is already available for the attribute based on the component-type alone, and which additional information may be gathered specifically for the quotation.

If the quoted price is accepted, then the repair shop needs to carry out the requested repair services, and as a consequence, the QA needs estimates of the costs of providing those services. To do so, the QA first estimates the costs per repair. These costs can be divided into the material costs and the allocated capacity costs for used work-hours and equipment. The former costs are the direct costs of performing the repair, while the latter costs are indirect costs.

The material costs consist mainly of the consumption of piece parts per serviced component; we denote those costs by the attribute *expected material costs* (EMC). Historical data on component workorders can serve as a basis to deduce a probability distribution on these costs, part costs exhibit substantial variability. By investigating the specific part-number, plane, component age, and operating conditions, a more precise estimate can be given but this is time-consuming.

Repair shops may reduce part costs by using so-called Parts Manufacturer Approval (PMA) parts. These parts are not produced by the original equipment manufacturer (OEM), but do have the same approved quality standards. Therefore, we introduce the attribute PMA that denotes the percentage cost reduction that can be achieved via PMA parts. This crucially depends on whether PMA parts are allowed at all within the requirements of the customer and within the regulatory requirements applicable for the customer. One may deduce the expected cost reduction from historical data for the component, but determining whether PMA parts are allowed is time-consuming. For a specific component type, the attribute PMA has only two likely outcomes, corresponding to the situations where PMA parts are or are not allowed, respectively.

A repair shop has limited capacity of (test) equipment and skilled maintenance engineers. But limited capacity can only be sold once. If the offered price is accepted, then this might affect the ability to bid on future RFQs. Hence, we interpret the utilization of time and capacity as opportunity costs and we introduce an attribute Capacity costs (Cap). Similar to the material costs, based on historical repairs it is possible to recover a probability distribution on the expected number of hours that resources will be used per component repair. To convert this to costs, the QA needs to estimate the opportunity costs per hour for those resources, based on current projected utilization.

A maintenance contract often has some agreements on the requested average repair turnaround time for the maintenance company. To ensure that these agreements are met, repair shops have a stock of so-called piece parts: Parts that are used in the repair. The investment and storage of such piece parts is costly, as unused piece parts at the end of the contract may need to be scrapped. To ensure that costs associated with these risks are appropriately reflected in quoted repair prices, we introduce an attribute SK, that defines the stock-keeping costs per year to attain the required level of service. Note that for large contracts with many repairs per year, this decreases the stock-keeping costs per unit.

An important factor that determines the likeliness that a tender is won, is the competition. The proposed model captures the impact of the competition by introducing a benchmark price, which reflects both the market price, the lowest price amongst all competitors, and any external factors such as the direct customer relationship and reputation. One could assume that a tender is won if the price offered by the repair shop is lower than the benchmark price, and lost if the offered price exceeds it. However, there will always be considerable remaining uncertainty even after the collection of this benchmark price. Moreover, only a rough estimate of market prices can be obtained by inspecting online marketplaces and past RFQs that were won and lost. Therefore, we model the benchmark price as a normally distributed random variable $BPri(\cdot,\cdot)$. We can collect information on the expected value μ of this variable, but we do not know the exact value of the best offer even after collecting case-specific information. By retrieving the attribute μ, we can deduce a probability of being the best offer and most attractive provider resulting in winning the contract. In our present model, we assume that the remaining standard deviation σ of demand after collecting case-specific information on the benchmark price is fixed. We could get a proxy for this by investigating past quoted prices, and whether they were accepted or not, but getting a precise estimate would be challenging.

The third and final factor that affects the value of the contract is the estimated total amount of work that will be carried out as part of it. This size is determined based on two important attributes. The first attribute, *period* defines for how many years the contract will run. This is often given in the contract, and otherwise it is relatively easy to estimate. Together with attribute Num, the number of repairs per year, we can determine the size of the contract. Note that although the client typically lists the expected number of repairs in the RFQ, there sometimes is a reason to question the client's estimate causing the need of more time to determine this attribute value. For both these attributes the initial estimates can be based on historical quotations. The size of the contract affects the effective overhead costs per repair, for example the stock-keeping costs, but may also affect the risk we are willing to take to win a contract: Large size contracts promise to be more profitable.

Hence, to describe the attributes in terms of the notation introduced in Section 4.3, we capture them in the set I of attributes, i.e.,

$$I = \{SK, Cap, EMC, PMA, Num, period, \mu\}.$$

4.4.4 Reward function

In line with the DIA model discussed in Section 4.3, the reward function expresses (in terms of the attributes) the expected monetary outcome for the repair shop when offering a certain quotation price. We next introduce and explain the reward function.

The most important part of the reward function is actually the costs of the repair. Given the introduced attributes for material costs (EMC), discount for use of PMA parts (PMA), capacity costs (Cap), we can determine the expected costs that are made per repair. It consists of the sum of EMC and Cap per unit repair. We subtract the possible percentage of discount for using PMA parts from the material costs EMC. Subtracting these costs from the quoted price gives the expected reward per repaired component if a price f is quoted, which we denote by ρ_f.

$$\rho_f = f - (s_{EMC} \cdot (1 - s_{PMA}) + s_{Cap})$$

Given that we know μ and σ for the $BPri(\cdot)$ random variable, it is possible to derive a probability of gaining the contract given the quoted price f. This probability is multiplied with the expected reward per component and the expected total number of repairs. Finally, the stock keeping costs are subtracted which results in the expected reward of the full maintenance contract, when awarded.

$$\overbrace{P(BPri(\mu,\sigma) \geq f)}^{\text{Quotation accepted}} \cdot \overbrace{s_{period} \cdot s_{Num}}^{\text{Contract size}} \cdot (\rho_f - \frac{s_{SK}}{s_{Num}})$$

If we let the reward consist purely of the potential reward of winning the contract, then the model tends to offer very high prices although it seems unlikely that the contract can be won. This might have an adverse effect on the image of the repair shop in the market. Therefore, if the likeliness of winning the contract is negligible, it is better not to place a quotation. To reflect this, let the decision space $\mathcal{F}$ consist of the union of a set of reasonable prices for the focal component type and the element no quotation.

To reflect the minor loss of image of placing a losing quotation, we multiply the probability of placing the losing quotation with a penalty costs reflecting the loss of image $\mathcal{I}$:

$$\overbrace{P(BPri(\mu,\sigma) < f)}^{\text{Quotation declined}} \cdot \overbrace{-\mathcal{I}}^{\text{Loss of Image}}$$

The complete reward function for placing a quotation $f \in \mathcal{F} \setminus \{\text{no quotation}\}$ is then obtained as follows:

$$r_f(s) = P(BPri(\mu,\sigma) < f) \cdot -\mathcal{I} + P(BPri(\mu,\sigma) \geq f) \cdot s_{period} s_{Num} \cdot (\rho_f - \frac{s_{SK}}{s_{Num}}),$$

Not quoting can never win the RFQ, but also avoids potential loss of image:

$$r_{\text{No quotation}}(s) \equiv 0$$

We note that while this reward function is a bit extensive, it is *conceptually* simple, straightforward and intuitive: Given the attributes, a simple spreadsheet may recommend a quotation price. The key difficulty for the QA lies in the sheer amount of RFQs that need to be treated, rendering the collection of all attributes for all RFQs intractable (cf. Section 4.1). The DIA model addresses this.

4.5. Numerical analysis

In this section, we discuss the numerical analysis that is performed for the Quotation Optimization model that was introduced in Section 4.4. In Section 4.5.1 we discuss, next to the optimal policy and two heuristic policies that can be used to solve the problem but simplify the comprehensibility of the policy. Section 4.5.2 introduces the information regarding the case study. The QO model and the collected parameters are based on a real airplane component at Fokker Services. In Section 4.5.3 we give the results of a full factorial experiment for this case and, in Section 4.5.4, we show and discuss the efficient frontier for the profit versus the time investment.

4.5.1 Policies

For QAs, sometimes it is hard to understand why specific actions are optimal in certain states considering the fully dynamic (optimal) policy. They follow the model for every quotation procedure and each time the order in which variables have to be retrieved could be different. In such an environment it is very hard for human decision makers, such as QAs, to fulfil their role as validator. Hence, there is a preference to build up a routine that does not require constant guidance by a decision support device. The optimal solution in this problem often requires such support.

Therefore, we investigate the actual gains of this flexibility compared to policies that could give routine to the QA. We introduce two policies that fix a part of the policy but still try to find a good trade-off between information acquisition costs and final reward. For the information acquisition process and the specific decisions that are taken dynamically, one can list three subdecisions in every state:

- The specific attribute that we retrieve if still acquiring information: $i \in I(s)$
- The subset of states where we switch from information acquisition to the final decision: $\tilde{S}^{\mathcal{D}} \subseteq \tilde{S}$
- The final decision that is taken after switching: $f(s) \in \mathcal{F}, s \in \tilde{S}^{\mathcal{D}}$

In this section, we discuss the differences between using a fully dynamic (optimal) way of information retrieval and two alternative policies. In addition, we use a naive policy as benchmark.

Fully dynamic policy (Policy 1)

Policy 1 has no restrictions on the allowed actions for a state. Using the stochastic dynamic program of Section 4.3.2, the optimal policy can be calculated meaning that in every state all three subdecisions that were discussed are optimized based on the probabilities of arriving in specific states.

Fixed-order policy (Policy 2)

We propose a policy that still tries to optimize two of the three subdecisions dynamically, but reduces the complexity of the solution by fixing the attribute retrieval order. We impose an attribute ranking such that there is always only one allowed subsequent attribute value to be retrieved. But, we still optimize in any state for the options of terminating the process by making a final decision or continuing for more information. Hence, we fix subdecision 1. As a consequence, the amount of flexibility in the first stage decreases, but still allows for the dynamics that we think can be crucial in these kinds of processes. In reality, the preferred ranking or order of attributes can be provided by the decision maker that has knowledge about the process. For this chapter, the attribute ranking is derived from the optimal policy.

For every attribute, we calculate the fraction of cases where this attribute is collected via the stochastic dynamic program (the fraction of final states $s \in S$, for which attribute i is collected). We rank these fractions, which can be seen as an indicator of importance.

Fixed-attributes policy (Policy 3)

The fixed-order policy is already simpler than the fully dynamic policy but still can contain sudden decisions that terminate the process, while being in the middle of the information acquisition phase. To further simplify the policy for the QA, we fix the order and number of attributes that is retrieved. Hence, this policy shows the effect of still optimizing the final decision while fixing the complete first stage of information acquisition, i.e., we fix subdecisions 1 and 2. This means that the ranking as used in the fixed-order policy is now incomplete and some attributes are therefore never retrieved. Actually, the specific order is not relevant anymore as we define a specific subset of attributes to retrieve. Let this subset be $I^* \subset I$. Then $\tilde{S}^{\mathcal{D}} = \{s \in \tilde{S} | \forall i \in I \backslash I^* : s_i = \perp, \forall i \in I^* : s_i \in R_i\}$. And, $\forall s \notin \tilde{S}^{\mathcal{D}}$, possible actions are in the set $\{i \in I^* | s_i = \perp\}$. We can check for different sizes of the set I^* what is the optimal final decision and calculate the expected reward. By fixing the optimal size of the set and the included attributes, the only decision to make is the final decision f. Note that this does not lead to a single variable optimization problem. The probability distributions of all remaining unknown variables are still taken into account, such that we maximize the expected profit. However, we lose much of the dynamics of the optimal policy. As a consequence, the QA can not be informed about possible different orders of retrieval for different appearing scenarios and is also not able to finish the process earlier in cases where extra information can be shown not to be relevant anymore. We use the same ranking that we defined for the fixed-order policy, which is the fraction of states for which an attribute value was acquired under the optimal policy. Note that we optimize this policy over the size of the attribute retrieval set $|I^*|$.

Naive policy

To benchmark the three previously discussed policies we introduce a naive policy. The naive policy can be seen as a specific instance of the fixed-attributes policy, where all attributes are collected before a final decision is taken: $|I^*| = I$. As such, we can always find the optimal final decision for every set of true attribute values. However, we do not save any costs by not retrieving the value of some variables, i.e., this policy is optimal in finding the maximum reward but is not likely to be optimal regarding profit.

In Table 4.3, we line up the different policies that we compare. In this table, i_s denotes the next action based on the predefined rank of attributes explained above.

Table 4.3: Policy comparison

Policy	Possible actions	
Fully Dynamic Policy	$I(s) \cup \mathcal{F}$	
Fixed-Order Policy	$i_s \cup \mathcal{F}$	
Fixed-Attributes Policy	$i_s \in I^*$	$\forall s \notin \tilde{S}^{\mathcal{D}}$
	$\mathcal{F}$	$\forall s \in \tilde{S}^{\mathcal{D}}$
Naive Policy	$I(s)$	$\forall s : I(s) \neq \emptyset$
	$\mathcal{F}$	$\forall s : I(s) = \emptyset$

4.5.2 Case Study parameters

In this section, we introduce a case study for one category of airplane components. The values that are chosen for this case study can be derived according to the description in Section 4.4. For every attribute we have taken exemplary values that fit an actual case of an airplane component category at Fokker Services. Note however that for computational tractability the distributions have been simplified to four possible values per attribute. Similarly, we have estimated the times to collect specific attributes based on a real-life case. Table 4.4 introduces the attributes' retrieval time τ_i, and their support R_i, where $\gamma = 50$.

We assume that all attribute random variables have a symmetric probability distribution over the support except for the Parts Manufacturer Approval variable PMA, which has a Bernoulli distribution. This symmetric distribution is defined by:

$$X \sim [\frac{p}{2}, \frac{(1-p)}{2}, \frac{(1-p)}{2}, \frac{p}{2}],$$

over a support of four values. The parameter p can be varied to create different distributions $(0 \leq p \leq 1)$. For example, if $p = 0.5$, we have a discrete uniform

Table 4.4: Attributes

i	R_i	τ_i	Definition
SK	{0,1000,2000,3000}	2	What are the total stock-keeping costs per year to attain the required service level?
Cap	{0,200,400,600}	10	What are the opportunity costs for using capacity and equipment in the repair shop?
EMC	{1200,1500,1800,2100}	14	What are the expected material costs per unit?
PMA	{0,0.1}	9	What fraction of the price can be saved by using alternative piece parts?
Num	{5,10,15,20}	2	What is the amount of repairs per year?
period	{1,2,3,4}	1	What is the duration of the contract in years?
μ	{1600,1800,2000,2200}	16	What is the μ parameter for the Benchmark Price?

distribution. Increasing the p parameter causes an increase in the variance, which will result in a more complicated optimal policy. The higher the total uncertainty in the model, the more information needs to be retrieved before a good final decision can be taken.

Although we use a similar distribution for all attributes except PMA, we would like to stress that the problem can still be solved for any combination of attributes and independent probability distributions. The distributions have small sample spaces to have a tractable state space and be able to perform the backward recursion and optimize the problem.

To terminate the process, a final decision f is needed for optimizing the reward function. For this example case we allow four possible values: $\mathcal{F} = \{1600, 1800, 2000, 2200\}$. We choose these four values based on the probability distribution for the benchmark price. When the QA knows that the average price in the market can be found somewhere in between 1600 and 2200, the QA will always try to put the quoted price in this same price range. Finally the costs of misjudging the benchmark value of a contract causing loss of image, $\mathcal{I}$, are set at 1500.

4.5.3 Full Factorial Test

To see what the effect is of different factors within the constructed case, such as the variance of the random variables, the market uncertainty for our specific case and the costs of retrieval, we conduct a full factorial experiment. We perform this experiment to determine the value of using a complex but optimal policy versus a suboptimal but easier to execute policy.

Table 4.5: Factorial analysis parameters

Parameter	Function	Values
c^*	Time multiplication factor	0.8,1,1.2
p	Uncertainty parameter	0.25,0.5,0.75
P_{PMA}	Probability of PMA parts	0.5, 0.7, 0.9
σ	Market uncertainty	50, 100, 200

Table 4.5 shows the parameters that were varied and their setup. We vary the impact of retrieval time, the uncertainty of the attribute distributions, the variance of the benchmark price, and the probability of having the PMA capability. The multiplication factor, c^*, is multiplied with all the $\gamma\tau_i$ to see the effect of different time settings. The effect of the increase of retrieval time is very important to study because this is what actually shows the value of the information that can be acquired. Second, the variance of the attributes is varied to see the effect, by varying the value of parameter p. The values of p are shown in Table 4.7b, together with the consequent standard deviation for attribute $Period$. Increasing the variance creates a larger spread of possible values, also increasing the value of knowing what the exact attribute value is. The last two parameters, P_{PMA} and σ, were chosen because of their strong effect within the model as can be seen in the reward function that was introduced. Combining these parameter values results in a set of 81 scenarios.

For every scenario and policy, we performed the full backward recursion to find the expected profit at the beginning of the process. The 81 individual scenario's can be looked up the in the Supplementary Materials of the paper Voorberg et al. [2023]. By fixing one parameter per row in Table 4.6, the gap to the naive policy of 27 different scenarios can be analyzed. The gap to the naive policy is the actual *profit improvement* because of the information acquisition optimization for the specific policy, which we denote by ϕ^{Policy}. This profit improvement can be maximally the sum of the retrieval costs of all attributes, shown in Table 4.7a. For the heuristic policies, the percentual gap compared to the maximal gap of the optimal policy is given. The fixed-attributes policy has an average profit improvement of 61% of the optimal policy's improvement and the fixed-order policy is reasonably close to the optimal policy with an average of 96% of its profit improvement.

If we compare the fixed-order policy with the fully dynamic optimal policy, there

Table 4.6: Full Factorial Average

	Value	$\phi^{Optimal}$	$\phi^{Fixed-Order}$	%	$\phi^{Fixed-Attribute}$	%	Naive Policy
c^*	0.8	**810**	761	*94*	404	*50*	9671
	1	**1092**	1042	*95*	648	*59*	9131
	1.2	**1404**	1362	*97*	975	*70*	8591
p	0.25	**1174**	1160	*99*	987	*84*	8552
	0.5	**1059**	1019	*96*	593	*56*	9178
	0.75	**1073**	986	*92*	448	*42*	9663
P_{PMA}	0.5	**1039**	983	*95*	583	*56*	8497
	0.7	**1093**	1046	*96*	655	*60*	9131
	0.9	**1174**	1136	*97*	788	*67*	9765
σ	50	**1092**	1049	*96*	654	*60*	9835
	100	**1091**	1049	*96*	667	*61*	9566
	200	**1124**	1066	*95*	706	*63*	7992
Average		**1102**	1055	*96*	676	*61*	9131

Table 4.7: Parameter insight

(a) c^*

c^*	Max. Cost Gap
0.8	2160
1	2700
1.2	3240

(b) Standard deviation for p

p	σ_{Period}
0.25	0.866
0.5	1.118
0.75	1.323

are cases in which it is actually better able to optimize the final reward. However, this additional reward does not weigh up to the additional costs for attribute value retrieval, i.e., the optimal policy always has the highest profit. Furthermore, the fixed-order policy outperforms the fixed-attributes policy in every scenario by a large amount (cf. Supplementary Materials). Hence, fixing the order of attributes has much less impact on the profit than also fixing the size of the set of attributes. When the cost multiplication factor c^* is increased, the gap between the optimal policy and the fixed-order policy actually decreases. We expect this to happen because of the increasing impact of the costs on the result, which is why in total the expected profit decreases. When retrieval costs are high, it is only valuable to retrieve a value if it has a significant impact in a large amount of states. Because of the design of the fixed-order policy, this means that both policies should converge to the same profit. To test this hypothesis, we performed an additional test where $c^* = 3$. In this case, we see that out of 27 cases in 11 cases there already is a 100% convergence in profit between the fixed-order and optimal policy. Similar behavior is observed for the fixed-attributes policy, although the convergence is much slower.

The uncertainty parameter p is showing a negative correlation with the effectiveness of the fixed-order policy compared to the optimal policy. The increase of uncertainty also increases the value of having information. Therefore, policies that are better able at finding which information is valuable perform better. Hence, both the fixed-order policy and fixed-attributes policy show a decrease in performance when the uncertainty increases. Furthermore, the increase of variance has a positive effect on the expected profit for the different policies. This effect can be explained by the capability of the policies to better distinguish between the 'extreme' cases of the distribution, which results in either higher rewards or higher losses. If attributes have values that affect the profit (extremely) negative, it is highly likely to make a loss if the request is not declined. Hence, it is easier to find out for the policies when to decline. Opposite, if attributes have values leading to high profits, the policies can more easily find out when to bid.

The probability of PMA is one of the crucial parameters in our current model. Therefore, what we see is that the more certain this attribute becomes, i.e., $P(PMA = 0.1)$ converges to either 0 or 1, the better the policies are able to predict the actual value of the attribute before retrieval and thus the reward increases. Furthermore, because of more certainty appearing in the model, we see that the heuristic policies again converge to the optimal policy.

The last parameter of the full factorial test is the standard deviation of the Market Price random variable. This random variable has a normal distribution and defines the probability of the Market Price being lower than the offer, in which case the deal would be lost and vice versa. An increase in uncertainty for this parameter should have a negative effect on the expected reward. We see this in the full factorial experiment, where the minimum profit decreases. However, the effect of information acquisition actually increases for the optimal policy. This is due to the increase of information value for attribute μ of the random variable. The larger spread of values because of a high σ increases the value of knowing the exact μ. It is important to see that the performance of the fixed-order policy is not really affected compared to the optimal policy, but the fixed-attributes policy does show a small percentual improvement in performance.

We can conclude that for all four variables in this full factorial test, the importance of information acquisition is shown and also the effects on the profit for those variables.

Every variable and parameter in this model has a direct effect on what is the optimal policy and the profit. The average gap between the fixed-order policy and the optimal solution is 46.83 or 4% of the optimal profit improvement, whereas this gap is 437.78 (39%) for the fixed-attributes policy. This means there is an average gap of less than 1 percent for the total profit between the optimal solution and the fixed-order policy. Hence, the crucial subdecision of the three that were introduced in Section 4.5.1 is to determine in which set of states a final decision is taken: $S^{\mathcal{D}}$. Meanwhile the first subdecision, the order of attribute retrieval, has no large effect on the profit optimization. Considering the value for a QA to know the fixed order of attributes that must be retrieved, which feels like a fixed routine, we conclude that the fixed policy would be a useful alternative for the fully dynamic policy as we introduced in this chapter.

4.5.4 Efficient frontier

The core of the information acquisition model is to optimize a trade-off between the use of time as a costly investment versus the maximization of reward as a result of this time investment. Given these two choices, one can build a Pareto frontier that maps the investment of expected time against the expected reward as a multi-objective optimization. Figure 4.1 shows the Pareto frontier for the specific example case that we introduced. In Appendix 4.A we describe the method of finding this frontier. It is important to stress that we optimize here for the average time invested for this specific model, i.e., one cannot set a hard constraint on the time spent for individual cases and expect the same reward. The value of this figure is that we show what is the trade-off between time and reward while not connecting the two through the costs of time. The optimal policy shows concave behavior between the reward and the invested time. The more time is invested the less value this time still has on increasing the expected reward. Furthermore, it is clear that the fixed-order policy performs very close to optimal in the whole range of invested time, which is a second indicator that it appears to be beneficial in the case of quotation optimization to use the fixed-order policy as a trade-off between optimality and comprehensibility. The fixed-attributes policy shows no concave behavior and is performing worse. Note that the naive policy is, like the other policies, visible in the far right point where the maximum amount of time is invested.

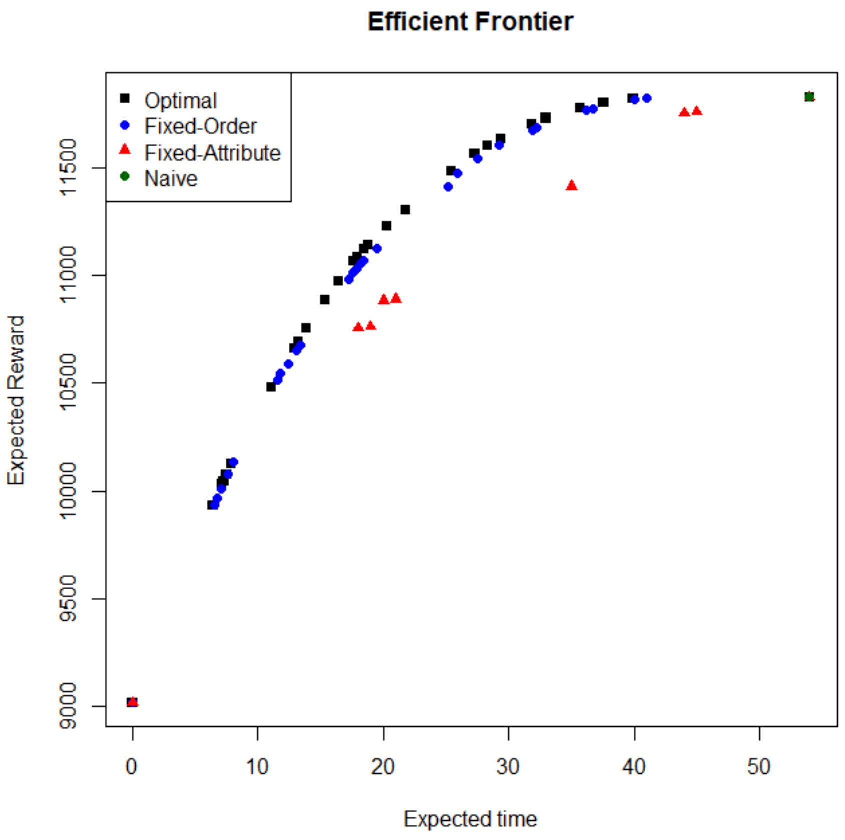


Figure 4.2: Efficient Frontier

4.6. Conclusion

We proposed a model for *dynamic information acquisition for profit maximization*, where there is a trade-off between investing time in acquiring more information and making optimal decisions. We introduced a formal mathematical solution of this DIA model and showed in an example what the effect of opting for more information can be on the expected profit.

For the specific problem of quotation optimization for independent contractors in aviation maintenance, we have introduced a refined version of the DIA model resulting in the QO model. We introduced a set of attributes that can affect the profitability of a maintenance contract and introduced a reward function that incorporates all these attribute effects.

We have shown that there are different levels of dynamism that can be used as a policy to increase comprehensibility and discuss the effect on the expected profit. We have shown that fixing the order of attribute retrieval has little effect on the expected profit. However, when also fixing the number of attributes that is retrieved, i.e., defining a set of attributes that is retrieved, we observed a strong decrease in expected profit.

In a case study based on an actual airplane component at Fokker Services, we have shown that indeed using a dynamic policy can save money compared to a more routine policy. Meanwhile, the loss of routine can annoy human decision makers. We showed that a middle way where the order of collecting attributes is fixed but the moment of making the final decision can be decided upon dynamically, gives a close to optimal solution.

Appendix

4.A. Efficient Frontier

The efficient frontier shows the maximum reward given a specific time that we want to invest, i.e., we have a first objective $f(s) = r_f(s)$ that is maximized with a second objective $g(s) = \sum_{i \notin I(s)} \tau_i$ that should be minimized. To calculate an efficient frontier we optimize the problem for a weighted linear combination of the two objectives. We vary the value λ which is a weight parameter that increases the weight of the acquisition costs. This results in different optimal policies where the amount of invested costs is valued against an optimal profit.

$$\mathcal{L}(s, \lambda) = f(s) - \lambda g(s)$$

Hence by optimizing this problem for different values of λ, we can find the optimal values for different time constraints.

4.B. Proof of Theorem 3.1

We have to prove that in state $(\perp, \perp)$ the value of retrieving attribute 2 as action is greater than or equal to the value of retrieving attribute 1 as action:

$$\int_{s_1} (V(s_1, \perp) - c_1) dF(s_1) \leq \int_{s_2} (V(\perp, s_2) - c_2) dF(s_2)$$

This is equal to showing that:

$$\int_{s_1} V(s_1, \perp) dF(s_1) - \int_{s_2} V(\perp, s_2) dF(s_2) \leq c_1 - c_2$$

Because $X_1 \sim X_2$, we can couple their distributions and rewrite this inequality as:

$$\int_{s_1} [V(s_1, \perp) - V(\perp, s_1)] dF(s_1) \leq c_1 - c_2$$

It holds that

$$V(s_1, \perp) = \max \begin{cases} \int_{s_1'} (s_1 + s_1') dF(s_1') \\ \int_{s_1} ((s_1 + s_1')^+ - c_2) dF(s_1') \\ 0 \end{cases}$$

$$V(\perp, s_1) = \max \begin{cases} \int_{s_1'} (s_1' + s_1) dF(s_1') \\ \int_{s_1'} ((s_1' + s_1)^+ - c_1) dF(s_1) \\ 0 \end{cases}$$

Hence

$$\int_{s_1} [V(s_1, \perp) - V(\perp, s_1)]\, dF(s_1) = \int_{s_1} \left[\max\left(\int_{s_1'} (s_1 + s_1') dF(s_1'), \int_{s_1'} ((s_1 + s_1')^+ - c_2) dF(s_1'), 0 \right) - \max\left(\int_{s_1'} (s_1' + s_1) dF(s_1'), \int_{s_1'} ((s_1' + s_1)^+ - c_1) dF(s_1'), 0 \right) \right] dF(s_1)$$

Since we subtract two max functions with the first and third term exactly equal, we only have to look at the second term. The maximal difference is obtained when both those second terms are larger than the first and third term. Hence

$$\begin{aligned}\int_{s_1} V(s_1,\perp) - V(\perp,s_1)dF(s_1) &\leq \int_{s_1}\left[\int_{s'_1}((s_1+s'_1)^+ - c_2)dF(s'_1) - \int_{s'_1}((s'_1+s_1)^+ - c_1)dF(s'_1)\right]dF(s_1)\\ &= \int_{s_1}(c_1-c_2)dF(s_1)\\ &= c_1 - c_2\end{aligned}$$

This completes the proof. □

4.C. Proof of Theorem 3.2

We have to prove that in state $(\perp,\perp)$ the value of retrieving attribute 2 as action is greater than or equal to the value of retrieving attribute 1 as action:

$$\int_{s_1}(V(s_1,\perp)-c)dF(s_1) \leq \int_{s_2}(V(\perp,s_2)-c)dF(s_2)$$

This is equal to showing that:

$$\int_{s_1} V(s_1,\perp)dF(s_1) - \int_{s_2} V(\perp,s_2)dF(s_2) \leq 0$$

Let $X_2^a \sim X_1 \sim N(0,\sigma_1)$. Let $X_2^b \sim N(0,\sqrt{\sigma_2^2-\sigma_1^2})$, such that $X_2 = X_2^a + X_2^b$, where X_2^a is independent of X_2^b. Redefine the state of our problem into $s = (s_1, s_2^a, s_2^b)$, where $s_2 = s_2^a + s_2^b$.

$$V(\perp,\perp,\perp) = \begin{cases}\int_{s_2^a}\int_{s_2^b}\int_{s_1}(s_1+s_2^a+s_2^b)dF(s_1)dF(s_2^a)dF(s_2^b)\\ \int_{s_1}(V(s_1,\perp,\perp)-c)dF(s_1)\\ \int_{s_2^a}\int_{s_2^b}(V(\perp,s_2^a,s_2^b)-c)dF(s_2^a)dF(s_2^b)\\ 0\end{cases}$$

But because $X_1 \sim X_2^a$, we can couple their distributions and rewrite this inequality as:

$$\int_{s_1} (V(s_1,\perp))dF(s_1) - \int_{s_2} (V(\perp,s_2))dF(s_2) = \int_{s_1} \left[V(s_1,\perp,\perp) - \int_{s_2^b} V(\perp,s_1,s_2^b)dF(s_2^b) \right] dF(s_1)$$

It holds that

$$V(s_1,\perp,\perp) = \begin{cases} \int_{s_2^b} \int_{s_1'} (s_1 + s_1' + s_2^b)dF(s_1')dF(s_2^b) \\ \int_{s_2^b} \int_{s_1'} ((s_1 + s_1' + s_2^b)^+ - c)dF(s_1')dF(s_2^b) \\ 0 \end{cases}$$

$$V(\perp,s_1,s_2^b) = \begin{cases} \displaystyle\int_{s_1'} (s_1' + s_1 + s_2^b)dF(s_1') \\ \displaystyle\int_{s_1'} ((s_1' + s_1 + s_2^b)^+ - c)dF(s_1') \\ 0 \end{cases}$$

Hence

$$\begin{aligned} &\int_{s_1} \left[V(s_1,\perp,\perp) - \int_{s_2^b} V(\perp,s_1,s_2^b)dF(s_2^b) \right] dF(s_1) \\ =&\int_{s_1} \left[\max \int_{s_2^b} \left(\int_{s_1'} (s_1 + s_1' + s_2^b)dF(s_1'), \int_{s_1'} ((s_1 + s_1' + s_2^b)^+ - c)dF(s_1'), 0 \right) dF(s_2^b) \right. \\ &\left. - \int_{s_2^b} \max \left(\int_{s_1'} (s_1 + s_1' + s_2^b)dF(s_1'), \int_{s_1'} ((s_1 + s_1' + s_2^b)^+ - c)dF(s_1'), 0 \right) dF(s_2^b) \right] dF(s_1) \end{aligned}$$

Note that $\int \max(f(x))dx \geq \max \int (f(x))dx$. Hence

$$\int_{s_1} V(s_1,\perp)dF(s_1) - \int_{s_2} V(\perp,s_2)dF(s_2) = \int_{s_1} \left[V(s_1,\perp,\perp) - \int_{s_2^b} V(\perp,s_1,s_2^b)dF(s_2^b) \right] dF(s_1) \leq 0$$

This completes the proof. □

5

Scalable algorithms for throughput time constrained decision-intensive processes

5.1. Introduction

Decision-making processes are becoming increasingly complex due to the exponential increase of information availability in this era. Environments where such processes appear on a daily basis are the financial sector, military tactics, politics, production planning, etc. Often these environments also have a direct time pressure to make the decision quickly. However, very complex processes can be overwhelming for decision makers, especially under such time pressure. This paper addresses the need for a decision-support solution in complex decision-making processes with throughput time constraints.

Retrieving an information attribute as part of a decision-making process may cost time. More specifically, the time spent on attribute retrieval can be measured in two different dimensions. First, we recognize the time that an employee invests directly, referred to as *working time*. The working time translates into the costs of utilizing a resource to finish the retrieval task. Second, in this chapter we introduce *lead times*. This is the time between the initiation of the retrieval and getting the result of that task. Lead time can be different compared to working time since the employee might not be directly involved the whole time due to waiting periods or use of external resources. Given these lead times, we may determine what we will refer to as the *throughput time*, which is the total lead time that passes between starting the process and making the final decision.

Large organisations have one advantage with regards to complex decision-making processes compared to small organisations: The number of people that can be involved in finding the best solution. More people are able to handle multiple things at the same time. Hence, tasks can be performed in parallel. This parallelism has two consequences for the optimization problem at hand. On the one hand, retrieving an attribute while other attributes are being retrieved decreases the marginal increase of the throughput time as a consequence of retrieving the task. Accordingly, the *marginal lead time* of a retrieval task refers to the effective time that this task adds to the throughput time. Note that this marginal lead time is case specific and depends on which other tasks are already being performed in parallel. Consequently, the decrease of this marginal lead time can result in a decrease of the throughput time, while still retrieving the same attributes. On the other hand, not being able to always wait for the outcome of a task weakens the information position when deciding on successive actions to take. In the previous chapters, all attributes values were either known or unknown whenever an action was taken. Our introduction in this chapter of parallelism and lead times results in a problem where some attribute values are *being retrieved* while a new action must to be taken, i.e., a third option is added to the list.

Allowing parallelism leads to a more challenging optimization problem. Indeed, parallelism allows for smart policies to divide the work in a way such that we can speed up the process while keeping a similar level of performance. Our model enables us to control both the average invested working time as well as the throughput time by performing tasks in parallel. By setting a maximum throughput time constraint, we aim to learn a policy that performs tasks in parallel to obtain a quicker response.

The inclusion of parallelism is also in line with how processes are modelled in the field of BPM. In Chapters 2 and 3, we already showed the value of using CMMN to model DIPs and how this approach can support the optimization of the process. The important assumption in these chapters is that tasks are instantaneously finished. This reduces parallelism to what is referred to as interleaving tasks, effectively disguising it by performing tasks subsequently. The assumption of instantaneous tasks therefore hampers the ability to support the full range of DIPs that can be modelled using CMMN. Dropping the instantaneous assumption enables us to model DIPs more accurately with CMMN. Subsequently, this allows the implementation and optimization of DIPs containing parallelism, which in turn underlies their usability in practice.

To optimize processes where important decisions have to be taken after having collected information, we introduced the Dynamic Information Acquisition model in Chapter 4. For large size problems where the complexity increases due to more information sources or more complex decisions, one cannot solve the DIA model exactly anymore. Moreover, the use of parallelism in the improved DIA model has major consequences. First, one needs to define how long it takes for each attribute to be collected, i.e., lead time next to working time. Second, since tasks can run in parallel, we need an extra variable that tracks the specific remaining collection time for each attribute that is being collected in parallel. The consequence is that the state space will grow enormously. Therefore, the need arises to find a method that can be used for optimizing large-scale DIA models where the state space exceeds the memory limits posed by computers.

Considering the large size problems and the complexity of the problem when introducing parallelism, it is not possible to solve the stochastic dynamic program (SDP) exactly anymore or to use the decomposition approach. Hence, in this chapter we propose an improved method that can approximately optimize these decision-making processes. Deep Reinforcement Learning (DRL) is widely considered to be an appropriate methodology for sequential decision-making problems that suffer from a state space explosion [Powell, 2019, Boute et al., 2021]. Reinforcement learning simulates the process to learn its characteristics in a similar way as one does in the case of policy iteration for an MDP. More specifically, the adoption of deep neural networks allows DRL algorithms to learn the best actions for any state in the state space, while not having to visit and test all states. In this chapter we propose a DRL algorithm for throughput time constrained DIPs. Furthermore, we improve the scalability of the algorithm by imposing a set of rules on the algorithm that are derived from four model-specific characteristics which are introduced in Section 5.3.3.

To test the performance of this new DRL algorithm, we show that the learned policy is close-to-optimal for a small example case. Moreover, we improve the quotation optimization (QO) model from Chapter 4 both in terms of size and via the inclusion of parallelism, in order to make it a more accurate representation of reality. This improved QO model can no longer be solved to optimality, but we demonstrate that our DRL algorithm yields a strong policy for this improved QO model.

Main contributions The main contributions of this chapter are as follows:

- We formulate a model that supports decision making while allowing tasks to happen in parallel, which enables decision making in environments with throughput time constraints.
- We improve the realism of the QO model, introduced in chapter 4, by introducing the notion of parallelism, and by including several relevant attributes that were not part of the original model.
- We demonstrate that a DRL algorithm that uses model-specific characteristics is able to solve the improved QO model.

The remainder of this paper is organized as follows. In Section 5.2 we give a positioning of this chapter in the thesis in reference to the other chapters and their respective section with related work. Section 5.3 introduces the Time-constrained DIA model that is used to solve our decision-making problem. Section 5.4 introduces the QO model with time constraints. In Section 5.5 we introduce the type of RL solution that is used for this model. In Section 5.6 we numerically study instances of a small case to compare the outcome of the DRL policy with the optimal policy. We also study a complex instance of the QO model where the DRL policy is compared to a base policy. Section 5.7 will conclude this chapter.

5.2. Position of work in thesis

We propose and study a quantitative model with time constraints and introduce a new solution method that contains reinforcement learning. Our contribution lies primarily in the new model and the introduction of a solution method that can withstand the increase in scale and complexity of the problem. We investigate the performance of this new DRL solution method against the optimal solution for small size problems. Furthermore, the introduction of a more complex version of the QO model allows to test the proposed solution method. For further literature on DIPs and the optimization of such processes we refer to the literature in Chapters 2 and 4. Since we are not aware of any publications on the use of RL to optimize DIPs, to place these contributions in a broader perspective we review applications of reinforcement learning in areas related to DIPs.

In Chapter 2 we introduced a new Optimizable DIP (ODIP) approach that helps to turn CMMN-modeled DIPs into an MDP model that can be optimized. Continuing on that, in Chapter 3 we have introduced a first approach towards the optimization of complex DIPs. This approach decomposes the large problem into a set of smaller ODIPs that are optimized exactly using the same approach as for the DIA model. However, the important assumption that must hold for this decomposition approach is the instantaneous retrieval of information. The solution is suboptimal and when introducing the possibility of performing parallel tasks, suddenly the decomposition becomes a setback that cannot optimize the parallel actions in multiple subprocesses while keeping in mind the throughput time constraint. This chapter introduces a second approach on how to optimize complex DIPs. This approach is specifically useful for decision-making processes in an environment with parallel information acquisition.

Chapter 4 introduced the DIA model that mathematically models a DIP and the SDP that solves such DIA models to optimality. That chapter shows that the SDP quickly runs to its limits in terms of scalability. Moreover, to assist a decision maker we show that a reasonable heuristic might work better while maintaining a high profit margin. Since this chapter's model contains parallelism, we cannot use the same heuristics to optimize the process. Therefore, we introduce a new DRL algorithm that aims to find a strong policy for optimizing DIPs. The throughput time constrained DIA model is an extension of the DIA model of Chapter 4.

5.3. The Throughput Time Constrained Dynamic Information Acquisition Model

In this section, we will extend the dynamic information acquisition for profit maximization (DIA) model of Chapter 4 to include a throughput time constraint as well as parallel attribute retrieval. Our description of the resulting model draws upon Chapter 4, but is self-contained for ease of reference.

For convenience we briefly mention the major changes with respect to the model in Chapter 4. We include lead times of tasks in stead of assuming instantaneous tasks, which allows to run tasks in parallel. Furthermore, we introduce the additional constraint of a maximum throughput time which is relevant in a model that has no instantaneous tasks. To include these changes in the model, we introduce a more complex set of variables that better keeps track of the status of each individual attribute.

5.3.1 The model

Consider a decision maker that needs to optimize a decision for a specific *case*. The goal is to select a single decision $f \in \mathcal{F}$. Suppose $\mathcal{F}$ contains finitely many alternatives. For every decision $f \in \mathcal{F}$, we define an expected reward (penalty) function $r_f(\cdot)$. For every decision $f \in \mathcal{F}$, this reward function depends on various parameters that denote the information attributes, indexed by $i \in I$, where I is the set of attributes.

At the start of the process, the value of the information attributes is uncertain. Learning the precise values for the case typically involves one (or multiple) employees studying the information of the case as well as related company information, which entails the investment of costly *working time*. In addition, learning the value of an attribute takes a certain amount of *lead time*. The decision maker's goal is to maximize the expected *profit* (which equals the expected reward obtained from the final decision net the costs incurred for retrieving attributes) while ensuring that a *throughput time constraint* is met. This constraint states that the cumulative *marginal lead times* between the start of the process and the final decision may not exceed the throughput time limit T. To model this throughput time limit, we keep track of the remaining time until the time limit is reached: This remaining time equals T at the start of the time frame allotted for the decision, and decreases subsequently, to $T-1, T-2, \ldots$, until it reaches 0 which corresponds to reaching the final limit of the time frame. The inclusion of this throughput time constraint may promote collecting multiple attributes in parallel to make more efficient use of marginal lead time (cf. Section 5.1).

The decision maker has access to the probability distribution for each parameter/attribute, denoted by the random variable $X_i, \forall i \in I$, all independently but typically not identically distributed. (See Chapter 4 for an extensive discussion on obtaining the distribution for parameters, and other matters pertaining to the practicality of modelling assumptions adopted here.) For notational convenience, we assume that all X_i have a discrete distribution.

The expected reward $r_f(\cdot)$ is a function of the true parameter values. To make this explicit, let $R_i := R(X_i)$ be the support of X_i, i.e. the set of values that can be taken on by attribute $i \in I$, and let $\mathcal{R} = R_1 \times \ldots \times R_{|I|}$ be the Cartesian product of support sets. Then for every $f \in \mathcal{F}$, $r_f : \mathcal{R} \to \mathbb{R}$. Furthermore, with slight abuse of notation, we will denote by $r_f(X_1, \ldots, X_{|I|})$ the reward of deciding $f \in \mathcal{F}$.

The problem is modeled as a sequential decision-making problem in discrete time. At each time point, based on the value of retrieved attributes, the decision maker can choose to initiate the retrieval of one or more additional attributes. We denote the number of time units needed to retrieve attribute i (lead time) by $\tau_i \in \mathcal{N}$, i.e. the value of attribute i is available τ_i units after its retrieval is initiated. For simplicity we assume that the working time that needs to be invested to retrieve attribute i equals θ_i, such that the retrieval costs equal $\theta_i\gamma$, where $\gamma > 0$ denotes costs per unit of working time. (Note that retrieving attributes in parallel reduces the marginal lead time required for attribute retrieval, but does not affect retrieval costs since the required working time is unaffected.) Apart from retrieving attributes, the decision maker can at any point decide to make a final decision, $f \in \mathcal{F}$, that ends the process. Since retrieved attributes enable the decision maker to make better final decisions, a trade-off arises between retrieving more attributes and making an early final decision.

At each time point, every attribute is in one of three different phases: An attribute is *known* when its precise value has been retrieved; after initiating the retrieval action but before the actual value has been retrieved, the attribute is *pending*; in all other cases, the attribute is *unknown*. The variable s_i keeps track of the status of attribute i: $s_i = \perp$ denotes that this attribute is unknown. When the attribute is pending, s_i must keep track of the remaining time until the retrieval process completes. To this end, P^t denotes the situation where an attribute i is being retrieved with t time units remaining until completion. When starting retrieval of attribute i, $s_i = P^{\tau_i}$, since it takes τ_i time units to collect the attribute. In general, while retrieving the attribute we have $s_i \in P_i$ with $P_i = \{P^1, ..., P^{\tau_i}\}$. A known attribute will be represented by the attribute value $s_i \in R_i$.

In general, the status of attribute $i \in I$ can be represented by $s_i \in \hat{R}_i$ with $\hat{R}_i = R_i \cup \{\perp\} \cup P_i$. Furthermore, define the *knowledge state* by $s = \{s_i\}_{i \in I}$. For any knowledge state s, define $I_\perp(s)$ to be the set of attributes with unknown values and not in process of retrieval, i.e. $I_\perp(s) = \{i \in I | s_i = \perp\}$. Similarly, denote the set of known attributes by $I_K(s) = \{i \in I | s_i \in R_i\}$ and the set of attributes being retrieved by $I_P(s) = \{i \in I | s_i \in P_i\}$. Denote by $I_P^c(s) = I_\perp(s) \cup I_K(s)$ the complement of $I_P(s)$, i.e. all attributes currently not being retrieved. Also, let the set of attributes which equal P^1 be $I_P^1(s) = \{i \in I | s_i = P^1\}$, and note that retrieval of these attributes will complete in the next time step. In general, the set of attributes which equal P^t will be denoted by $I_P^t(s) = \{i \in I | s_i = P^t\}$. Define the set of knowledge states by $\hat{S} = \hat{R}_1 \times \hat{R}_2 \times \ldots \times \hat{R}_{|I|}$. At any point in time, the process state is given by $\sigma = \{s, t\}$, with $s \in \hat{S}$ and $t \in \{0, 1, \ldots, T-1, T\}$, i.e. the state consists of the knowledge state, and the remaining time t until the available throughput time runs out.

The expected reward associated with each final decision $f \in \mathcal{F}$ is expressed as a conditional expectation. In particular, for any knowledge state s, this reward equals

$$\tilde{r}_f(s) := E[r_f(X_1, \ldots, X_n) | \forall i \in I_K(s) : X_i = s_i] \tag{5.1}$$

For defining the set of possible actions $A \in A(s)$, any final decision will be denoted by $f \in \mathcal{F}$. The action of initiating retrieval of some unknown attributes is denoted by a nonempty subset $A \in I_\perp(s)$. The decision to wait one time step, i.e. to neither retrieve additional attributes nor make a final decision, can be conveniently denoted by the empty set $\emptyset$. The set of possible actions in any time step is therefore the union of the power set of unknown attribute values and the set of final decisions, i.e. $A(s) = \mathcal{P}(I_\perp(s)) \cup \mathcal{F}$, where $\mathcal{P}(\cdot)$ denotes the power set.

We are now ready to model the problem as a stochastic dynamic program. Suppose that the knowledge state after taking some action $A \in \mathcal{P}(I_{\perp}(s))$ is $s = \{s_i\}_{i\in I}$. Then the probability $P(s,s')$ to start the next period with knowledge state $s' = \{s'_i\}_{i\in I}$ can be expressed as follows:

$$P(s,s') = \begin{cases} \prod_{j\in I_P^1(s)} P(X_j = x'_j) & \text{if } \forall i \in I_P^1(s) : s'_i = x'_i, \\ & \text{and if } \forall t > 1 \forall i \in I_P^t(s) : s'_i = P^{t-1}, \\ & \text{and if } \forall i \in I_P^c(s) : s'_i = s_i \\ 0 & \text{otherwise} \end{cases},$$

Note that when $I_P^1(s) = \emptyset$, we obtain $P(s,s') = \prod_{j\in\emptyset} P(X_j = x'_j)$, which we define to be 1.

Now, let $V(s,t)$ denote the expected optimal profit incurred until the end of the process when starting in knowledge state $s \in \hat{S}$ with t time periods available until the available throughput time runs out. $V(s,t)$ can be computed via the following equations that hold for any $s \in \hat{S}$ and any $t > 0$:

$$V(s,t) = \max\{\max_{A\subseteq I_{\perp}(s)} \hat{V}(s,t,A), \max_{f\in\mathcal{F}} \tilde{r}_f(s)\}, \tag{5.2}$$

$$\hat{V}(s,t,A) = \tilde{V}(s',t) - \sum_{i\in A} \gamma\theta_i, \qquad \text{with } s' \text{ s.t. } \forall i \in A : s'_i = P^{\tau_i}, \forall i \in I \setminus A : s'_i = s_i \tag{5.3}$$

$$\tilde{V}(s,t) = \sum_{s'\in\hat{S}} P(s,s')V(s',t-1), \tag{5.4}$$

$$V(s,0) = \max_{f\in\mathcal{F}} \tilde{r}_f(s), \tag{5.5}$$

5.3.2 Example instance

To illustrate the model introduced in Section 5.3.1, we next introduce an example instance. We consider four information attributes with associated random variables X_1, X_2, X_3, X_4. The possible final decisions are $\mathcal{F} = \{100, 600, -100, -600\}$, and the reward function is $r_f(s) = (s_1 + 2s_2 + 3s_3 - 2s_4) \cdot f$. The attributes take on values in the set $\{0,1,2,3\}$. We set the lead times for retrieval as $(\tau_1, \tau_2, \tau_3, \tau_4) = (5,6,6,5)$ and assume that the working times are equal: $\tau_i = \theta_i$.

We let the maximum throughput time be $T = 14$, and we let retrieval costs be equal to the calendar retrieval times, i.e. we set $\gamma = 1$. Since sequentially collecting all attributes would take 22 time units, collecting all attributes is only feasible when parallelism is employed. Hence, a trade-off arises both on the order of information collection and the impact this has on the information position to make the final decision.

To illustrate the recursive expressions of Section 5.3.1 and associated notation, suppose that the first action is to collect attributes 2 and 4. Then:

$$V(\perp,\perp,\perp,\perp,14) = \hat{V}(\perp,\perp,\perp,\perp,14,\{s_2,s_4\}) \tag{5.6}$$

$$\hat{V}(\perp,\perp,\perp,\perp,14,\{s_2,s_4\}) = \tilde{V}(\perp,P^6,\perp,P^5,14) - \gamma(6+5) \tag{5.7}$$

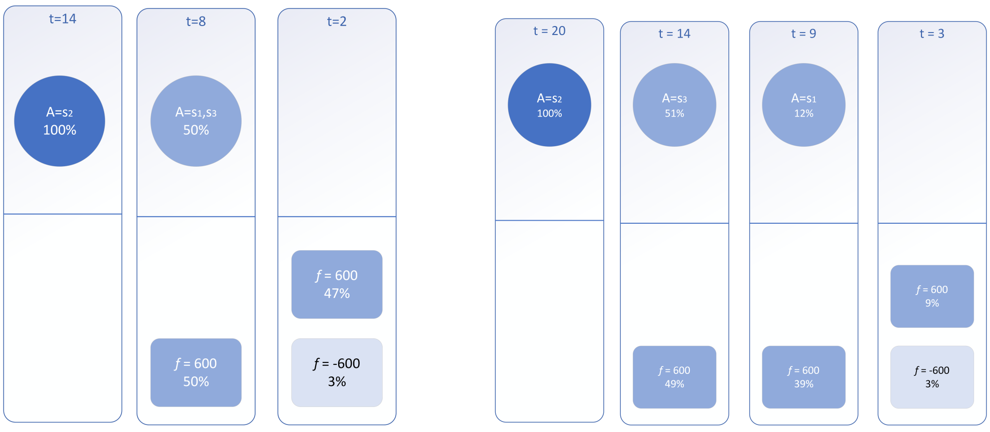
$$\tilde{V}(\perp,P^6,\perp,P^5,14) = V(\perp,P^5,\perp,P^4,13) \tag{5.8}$$

This example instance is relatively small-scale, and hence we are still able to evaluate the recursive expressions to find the optimal policy, which turns out to correspond to an expected profit of 3626.35. To illustrate the policy, we initiate the process 10000 times and use Monte Carlo simulation to trace the process until a final decision is made. The results are visualized in Figure 5.1a. The percentages denote the percentage of 10000 cases that followed this policy. To illustrate the impact of the throughput time constraint, we similarly visualize the optimal policy for a maximum throughput time $T = 20$ in Figure 5.1b. This latter policy corresponds to an expected profit of 3628.68, demonstrating that an increased throughput time already enables a reduction of attribute retrieval costs for this basic example.

The modest size of our example results in a tractable policy; for larger-scale models, the use of parallel tasks will cause large state spaces to appear with more complex policies as a result.

5.3.3 Model properties

While our model allows all actions $A \in A(s)$ to be taken in knowledge state s, intuitively the action space can be substantially reduced by imposing some properties that we would expect to hold true for the optimal policy. We next list these properties, and briefly discuss our reason for imposing them. In Section 5.5, we will leverage these properties to improve the convergence of our DRL algorithm for the problem.



(a) Maximum throughput time $T = 14$. *(b)* Maximum throughput time $T = 20$.

Figure 5.1: Visualization of policies for the example instance.

The properties are presented as conjectures that we expect to hold for the throughput time constrained DIA model with at most mild additional assumptions:

Conjecture 1. *If the retrieval time τ_i for an attribute i exceeds the remaining time t before the remaining throughput time runs out, then it is not optimal to retrieve attribute i.*

This first conjecture can be reasoned logically. If one would decide to commence with retrieving an attribute which won't be there before the throughput time runs out, it means they have to incur the corresponding costs for this task, while the resulting attribute value will never arrive before a final decision must be made. Hence, the information position won't improve while costs are being made. This is never an optimal policy.

The second property is a bit more involved:

Conjecture 2. *It is optimal to collect attributes only in the first time period $t = T$ (i.e. with T time periods remaining), and in periods where the value of attributes that were collected earlier are revealed. (More precisely, there exists an optimal policy that satisfies this property.)*

Indeed, if we decide to retrieve an attribute in a period $t < T$, where no information was revealed in period t, i.e., the information position did not change, then we could equally well have started retrieval of that attribute in an earlier period. This leaves us more time later in the process to collect other important information.

A final conjecture deals with the situation where all attributes have been collected:

Conjecture 3. *There exists an optimal policy that takes a final decision whenever all attribute values have been collected.*

We present no formal proof of these properties, but believe they can be proven rigorously via an induction argument based on dominance arguments. In the remainder of this chapter, we will impose these properties whenever searching for (near-)optimal policies.

5.4. Quotation Optimization Model with time constraints

In this section we introduce the basic problem and the model that we use to study the time-constrained DIA model: the quotation optimization model with time constraints. The problem is largely inspired by the QO model introduced in Chapter 4. The QO model represents a quotation process for an aircraft component. Different instances of the model can be used for different aircraft component types. In practice, such models would be constructed drawing from information of previous quotation processes and repairs involving the same component type (cf. Chapter 4).

5.4.1 Process

A quotation officer seeks to optimize the quotation price for the maintenance of a certain airplane component based on an RFQ. Several information attributes may improve the officer's knowledge on the best price to quote. This information is specific for each individual RFQ.

However, collecting attributes costs working time as well as lead time, which leaves the quotation officer in a complex net of trade-offs with regards to throughput time versus total working time versus the quality of the offered quotation price. To achieve the best possible price, the quotation officer would need to know the information of all attributes. On the other hand, the collection of information costs working time and since we try to maximize the profit of the deal it is preferred to find the correct price without collecting too much costly information. To avoid collecting attributes that turn out not to be needed, it would be best to collect attributes sequentially (cf. Chapter 4). However, a throughput constraint requires the officer to come back with a quote within a certain time frame to be competitive in the market. Hence, some information might have to be collected in parallel to quickly acquire enough information to be in the right information position. Therefore, the decision on which information needs to be collected when becomes even more precarious. We seek to manage the trade-off between cost reduction and revenue maximization, by collecting valuable information and afterwards quoting an appropriate price, all within an acceptable throughput time.

5.4.2 Attributes

A repair shop for aircraft components can use a variable set of attributes to come to a result. Collection of several information attributes involves a question that is asked by the quotation officer employees of the repair shop that will handle the repair. In line with our focus on parallelism, employees of the repair shop can perform such tasks while the quotation officer focuses on something else.

Table 5.1: Attributes (SK = Stock-Keeping, MaC = Machine hour Cost, MH = Machine Hours, CUtil = Component Under-utilization, EMC = Expected Material Cost, PMA = Parts Manufacturer Approval, Num = Number of repairs per year)

i	R_i	τ_i	Definition
SK	{0,500,1000,1500 2000,2500,3000}	2	What are the total stock-keeping costs per year to attain the required service level?
*MaC**	{0,20,40,60,80,100,120}	4	What are the hourly costs of using the set of machines in the repair shop for this item ?
*MH**	{0,1,2,3,4,5,6}	6	How many hours are needed on these machines for one item?
*CUtil**	{1,2,3,4,5,6,7}	7	What is the risk of under-utilization of the set of machines corresponding to the component if the quotation is not won?
EMC	{1500,1800,2100,2400}	14	What are the expected material costs per unit?
PMA	{0,0.05,0.1,0.15,0.2}	9	What fraction of the price can be saved by using alternative piece parts?
Num	{5,10,15,20,35,30,35,40}	2	What is the amount of repairs per year?
period	{2,4,6,8}	1	What is the duration of the contract in years?
μ	{1600,1850,2100,2350 2600, 2850, 3100}	16	What is the μ parameter for the Benchmark Price? (normally distributed random variable)

Our QO model extends the model introduced in Chapter 4. It contains nine attributes, the majority of which have already been introduced in Chapter 4. However, to increase the realism of the model while enabling us to demonstrate the scalability of the proposed algorithms, we introduce three additional attributes (denoted with a *) that together represent the opportunity costs of using capacity in the repair shop. (This opportunity costs was directly represented by a single attribute (s_{CAP}) in Chapter 4; the present approach enables us to more granularly represent the process of estimating this opportunity cost.)

The opportunity costs in the workshop can be viewed at as the costs that are made for taking away capacity for future opportunities of repairs. These opportunities might be more valuable and hence the current RFQ might be less interesting. These opportunity costs for a single job in the RFQ depend mainly on three variables: 1) What are the capacity costs of using the set of machines in the workshop for this item (MaC), 2) How many hours are needed on these machines for one item (MH), 3) What is the probability that these machines will remain underutilized if we do not win this quotation (CUtil). The opportunity costs for putting this order on the capacity of the workshop mainly depends on the current workload without the new order. If the workload of the necessary machines is already high (which corresponds to a low probability of under-utilization), then increasing the workload results in less available opportunities in the future. Moreover, machines with a very high workload are prone to being less efficient causing more production errors. Therefore, a high risk of under-utilization effectively decreases the workshop's opportunity costs. To conclude we charge an hourly costs multiplied by the number of expected hours that are needed for the RFQ in the workshop. To account for the underutilization risk, we divide by the square of this risk factor. Hence, the capacity opportunity costs can be defined by:

$$s_{Cap} = \frac{s_{MaC} \cdot s_{MH}}{s_{CUtil}^2}$$

Table 5.1 contains all the information attributes that are used for the QO model.

5.4.3 Reward function

The optimization objective of the process is to maximize the reward of the final decision while minimizing information acquisition cost. The demanded throughput time is a hard constraint that cannot be ignored. In this section we introduce the reward function that contains all the information attributes as input parameters and has the final decision as a decision variable. Decreasing the number of uncertain input parameter values will decrease the uncertainty of the outcome of the reward function.

The reward function is built based on an expected reward per repaired component, multiplied by the total amount of components in the contract. The expected reward per repaired component consists of the bid value per repair minus costs such as material costs (possibly discounted for alternative piece parts) and the workshop opportunity costs as explained in the previous section. Let ρ_f be the expected reward per repaired component if a price f is quoted per component:

$$\rho_f = f - (s_{EMC} \cdot (1 - s_{PMA}) + \frac{s_{MaC} \cdot s_{MH}}{s^2_{CUtil}})$$

Subsequently, some overhead costs are subtracted and a benchmark price is determined that represents the value for which the contract is won in the market. This results in a reward function:

$$r_f(s) = P(BPri(\mu, \sigma) < f) \cdot -\mathcal{I} + P(BPri(\mu, \sigma) \geq f) \cdot s_{period} s_{Num} \cdot (\rho_f - \frac{s_{SK}}{s_{Num}}),$$

The BPri variable represents the uncertain benchmark price in the market that determines if the RFQ bid beats the competitors. It is modeled as a normal random variable with μ as one of the uncertain attributes and σ fixed. If the bid is lost we incur a costs $\mathcal{I}$ denoting the loss of image in the market of quotation optimization. In case that the bid is won, the expected reward per repaired component is multiplied by the amount of component repairs in the contract minus any stock-keeping costs to attain the conditions of the bid. The result is the profit over the full contract.

5.5. DRL Algorithm

We propose a DRL algorithm for solving the Throughput Time Constrained DIA model. For our algorithm, we modify and improve the *deep controlled learning* (DCL) algorithm [van Jaarsveld, 2020] specifically for the DIA model. DCL adapts classification-based DRL algorithms (Lagoudakis and Parr [2003]; see also Silver et al. [2017]) and is especially suitable for highly stochastic environments. In this section, we discuss its application to the DIA model: Our exposition focuses on the modifications and improvements that were made in order to succesfully apply DCL to this model.

5.5.1 MDP reformulation

In Section 5.3.1, we saw that the action space for the Throughput Time Constrained DIA model contains the powerset of $I_\perp(s)$: In each period one may select any combination of unknown parameters for retrieval. The large numbers of actions in this powerset renders it cumbersome to represent policies as neural networks, which complicates DRL.

To overcome this issue and enable the use of neural network policies for the model proposed in this paper, we transform the action space and MDP into an equivalent model with a smaller action space. In particular, for the equivalent model the selection of attributes for retrieval in each period becomes a sequential process for each period: In each step at most a single attribute is selected. Selecting attribute s_i for retrieval in period $t > 0$ brings the attribute state to P^{τ_i}, but leaves the remaining time at t which enables the selection of additional attributes for retrieval this period. Only after selecting the *pass* action $\emptyset$, we advance the time by 1 period. So for knowledge state s, the action set becomes $A(s) = I_\perp(s) \cup F \cup \emptyset$.

This results in the following recursive equations that hold for any $s \in \hat{S}$ and $t > 0$:

$$V(s,t) = \max\{\max_{a \in I_\perp(s)} \hat{V}(s,t,a), \max_{f \in \mathcal{F}} \tilde{r}_f(s), \tilde{V}(s,t,\emptyset)\}, \tag{5.9}$$

$$\hat{V}(s,t,a) = V(s',t) - \gamma\theta_a, \qquad \text{with } s' \text{ s.t. } s'_a = P^{\tau_a}, \forall i \in I \setminus a : s'_i = s_i \tag{5.10}$$

$$\tilde{V}(s,t,\emptyset) = \sum_{s' \in \hat{S}} P(s,s')V(s',t-1), \tag{5.11}$$

$$V(s,0) = \max_{f \in \mathcal{F}} \tilde{r}_f(s), \tag{5.12}$$

One may verify that any policy for the original MDP can be transformed into an equivalent policy for this equivalent MDP, and that the two policies will result in the same costs. However, while representing policies for the original MDP as a neural network is cumbersome, our transformed MDP has a much more manageable action space which directly supports neural network policies. Therefore, we overcome the difficulties of the large action spaces that feature in our original model formulation (see §4.3 of Boute et al. [2021] for an extensive discussion of the difficulties of applying DRL for large action spaces).

5.5.2 Custom sampling of states

DCL training performance, like any DRL algorithm, is highly sensitive to the states that can be sampled during training. For finite-horizon models such as the Throughput Time Constrained DIA model, DCL samples states that it encounters while running (an improved version of) the current policy, when starting from the initial state for the problem. This initial state corresponds to all attributes being unknown, while the remaining time until the available throughput time runs out initially corresponds to T.

However, always starting from the same state leads to little exploration and generalization since we quickly learn in which specific direction to go within the state tree, implying that we do not learn effective actions in other areas of the state tree (and hence have little incentive to visit those areas).

To force the agent to learn also in unknown areas we adopt random initial states while training. Initial states first of all randomize the initial knowledge state. In particular, half of the attributes is assigned a random remaining time of retrieving the value of the attribute. Hence, the starting process already has some attributes in the process of being collected. The other half of the atttributes is either assigned an already retrieved value or remains an unknown attribute. Moreover, for the initial state the time t remaining until a decision is forced, is randomized and also the throughput time constraint value T is varied such that the agent learns how to deal with different amounts of remaining time.

5.5.3 Optimization for various throughput times

The remaining throughput time T is a key MDP parameter. Training an appropriate policy for different throughput times T in principle involves retraining the policy for each new MDP. However, by including the remaining time t until the available throughput time runs out as a neural network feature, and by randomizing the initial value t during training (as discussed in Section 5.5.2), we may learn and use a single neural network policy for multiple MDPs with different maximum throughput times T. In the scenario where a company decides on the throughput time based on the number of arriving process instances, e.g., RQFs, it is very useful that the model can adapt easily to a change of maximum throughput time.

Leveraging structural properties

Based on the three structural properties that we derived in Section 5.3.3, we can further simplify the architecture of our model. For example, by not allowing the agent to take an action when there is no new information arriving, we decrease the amount of allowed actions in a large set of states by a significant amount, greatly simplifying the task of identifying an well-performing policy. Similarly, by not allowing to take actions when they take longer than the remaining throughput time, we decrease the complexity of the problem that should be solved, and thus simplify the task of learning a good policy.

5.5.4 Common random numbers

van Jaarsveld [2020] discusses the use of common random numbers to improve the quality of generated policies. However, van Jaarsveld [2020] showed that any events that influence costs and transitions can be modeled to occur independently from the actions taken in the model. In the DIA model, attributes are revealed based on the actions taken, thus ruling out the common random numbers approach as introduced in van Jaarsveld [2020]. However, since the randomness of any sequence of states and actions can be fully captured by the values of the attributes that are to be revealed when retrieved, we may successfully adapt the approach proposed in van Jaarsveld [2020] to our model. This vastly improves the learning convergence.

5.6. Numerical analysis

In this section we show how well the DRL algorithm discussed in Section 5.5 performs compared to alternative heuristics. For the small example of Section 5.3.2 we benchmark with a base heuristic that will be explained and the optimal policy obtained via backward recursion. For the QO model with time constraints we can only compare with the base policy.

We next detail the choice of DRL hyperparameters for our experiments [cf. van Jaarsveld, 2020]. We train 10 generations of neural networks for each problem instance of the small example and 15 generations for the QO model case, and report performance of the best-performing one. We adopt multi-layer perceptron networks with four hidden layers with dimensions 128, 64, 64, and 64, respectively, which are trained using a minibatch size of 64. For training each network, we employ $K = 50000$ and 500000 samples for the example case and QO model case respectively, with $\underline{n} = 500$, $\bar{n} = 4000$, $\epsilon = 0.05$ and $\beta = 0.2$.

5.6.1 Example case

For the small example that was introduced in Section 5.3.2 we have performed a full factorial comparison based on three parameters:

- Throughput time T
- The cost factor γ
- Distribution variability

With the throughput constraint parameter we look at the impact of this parameter on the profit. A less tight throughput time constraint should allow for better and more complex policies that result in higher profits. The cost factor γ parameter is used to examine the sensitivity of the policy to increasing attribute retrieval costs. Higher acquisition costs require a higher return in value when acquiring an information attribute. Therefore, changing these cost impacts the policy behavior. Finally, the variability parameter denotes three scenarios that all create different distributions over the possible values for each information attribute. In Appendix 5.A, one can find the three different distributions for each information attribute.

The results of our experiments can be found in Table 5.2. The rows of the table correspond to the average performance for various problems that share a certain common parameter value, i.e. the first row corresponds to the 9 problems that each have $T = 17$. Averaged over those problems, each row contains the average optimal costs retrieved via recursive evaluation of the value function via the equations discussed in Section 5.3.1. The row also contains the performance of the neural networks policy trained using the algorithm discussed in Section 5.5. Finally, we compare these two policy costs to the objective value corresponding to a base policy that does not retrieve any attributes but instead directly takes the final decision that is optimal in expectation. We also tabulate the performance gap of this latter policy to the other two policies.

Table 5.2: Full Factorial experiment based on the small example in Section 5.3.2. The BASE column shows the average profit for 9 different scenarios for the BASE policy. The next three columns show the gaps between the profits of the different policies.

Average costs		**BASE**	**DRL - BASE**	**OPT - BASE**	**OPT - DRL**
Throughput time T	**17**	3600.8	224.9	235.3	10.4
	14	3597.3	237.8	239.3	1.4
	11	3593.8	234.6	236.6	2.0
Cost Parameter γ	**0.75**	3597.3	230.5	240.7	10.2
	1	3597.3	235.5	236.8	1.3
	1.25	3597.3	231.4	233.7	2.3
Variability Scenario	**0**	3594.2	33.0	35.3	2.3
	1	3598.7	230.4	232.2	1.8
	2	3599.0	434.0	443.6	9.6

Regarding the base policy, we can see that it behaves independently of the acquisition cost parameter. Since this policy directly makes a final decision, acquisition cost are never incurred. For the variability and the throughput time constraint there are small differences that are caused by the variability of the problem which impacts the learning slightly. Both the DRL solution and the optimal solution make a significant improvement compared to the base policy. Indeed, between these two the gap is almost negligible. However, since one of the 27 cases ($T = 17, \gamma = 0.75$, variability level 2) has a performance gap of 70, values corresponding to these three parameters are showing a bit larger gap. Hence, at least for this example case we can conclude that the DRL solution is a very promising method that is able to find close-to-optimal policies for many instances.

The three parameters have a clear impact on the optimal policy behaviour. It was shown in Section 5.3.2 that the throughput time constraints can have an impact on the policy. Similarly, the variability of the random variables has an impact on how much impact it has on acquiring an attribute value.

5.6.2 QO Model with time constraints

In this section we discuss the results and outcomes of the numerical tests that have been performed on the QO model with time constraints. Similar to the example case we have performed a full factorial test to compare the DRL solution to a base solution. However, due to the complexity of the problem it is not possible to make a comparison to the optimal solution. Next to the full factorial test we have optimized the problem for the full range of possible throughput time constraint values. Using this, we can show the profit improvement as the throughput time constraint becomes more lenient.

For the full factorial test we have used the same parameters as for the small example: throughput time limit T, acquisition cost γ and variability levels are varied. The results can be found in Table 5.3. Although the DRL solution performs significantly better compared to the benchmark, there are some surprising results that can be mainly explained due to the large number of random variables in the problem that impact the algorithm. For example, the DRL result in the test is not monotonically increasing based on throughput time, though deviations are very small. This might have to do with some variability in DRL performance, as we also saw for the smaller problem. However, since the DRL solution significantly outperforms the benchmark, it is clearly able to learn effectively which information attributes to retrieve in order to improve performance.

Table 5.3: Full Factorial experiment based on QO model with time constraints. The BASE column shows the average profit for 9 different scenarios using the BASE policy. The next column shows the gap between the BASE policy and the learned DRL policy.

Average costs		**BASE**	**DRL - BASE**
Throughput time	**22**	5083.3	3411.2
	32	5027.8	3042.2
	42	4920.6	3336.7
Cost parameter	**0.75**	5017.2	3738.6
	1	4965.6	2953.6
	1.25	5048.9	3097.9
Variability Scenario	**0**	3298.1	1225.3
	1	1474.2	1083.6
	2	10259.4	7481.1

In Figure 5.2 we focus on how well the DRL policy can adapt to a changing maximum throughput time T, while fixing the cost parameter $\gamma = 1$ and the variability scenario to 0. Note that we assume the throughput time parameter as a given constraint, which cannot be easily changed. As discussed in Section 5.5, we train and test a single neural network policy for various values of T, by randomizing the initial state. As expected, the benchmark policy consistently attains the same performance since it always makes the same final decision without retrieving attributes. Furthermore, we see that the DRL policy successfully adapts to various values of T, making use of the increasing freedom in sequentially retrieving various policies as the throughput time constraints becomes more lenient. The sudden increase of profit at time step 16 is because information attribute μ takes 16 time steps to collect. Hence, only with a throughput time of more than 16 time steps is it possible to collect this information. Around 40 time steps, we reach the maximal profit for the DRL solution (with small fluctuations). Since the sum of all attributes equals 61, it is not surprising that after 40 the optimal solution is found: Not all attributes are cost-wise needed and so if all other attributes can be collected sequentially due to the large allowed throughput time, the result will not change anymore. We can conclude that the DRL algorithm is good in adapting to situations where a new throughput time constraint is used.

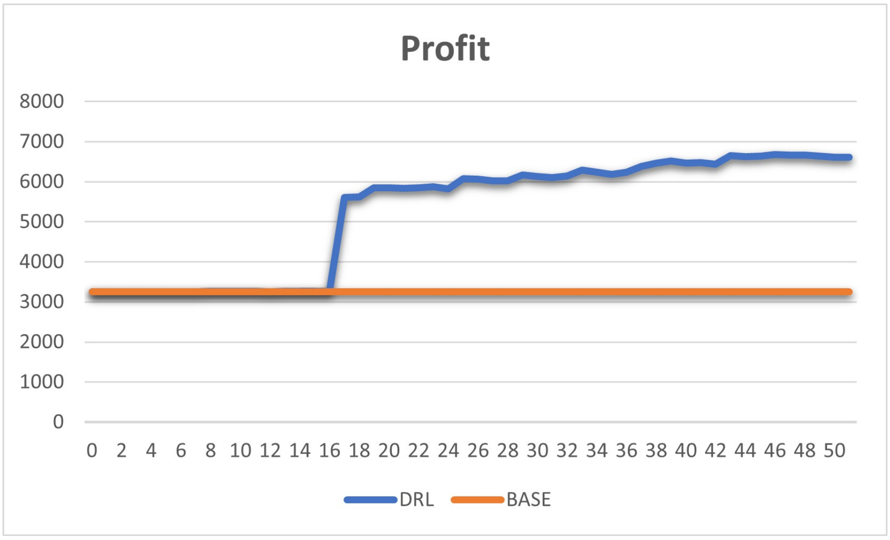


Figure 5.2: Profit increase based on throughput time constraint

5.7. Conclusion

The increasing amount and complexity of decisions due to the increase of available information calls for methods to support human decision makers in optimizing decision-making processes. Moreover, due to the size and vast amount of processes there is the need to be able to set time limits on how long to take before a decision must be taken. In this chapter we have introduced a *Throughput Time Constrained Dynamic Information Acquisition Model*. The model optimizes a decision-making process consisting of an information acquiring phase and a decision-making phase. For the information acquiring phase we define a throughput time constraint that sets a maximum time. By allowing information attributes to be collected in parallel, it is still possible to optimize these processes not only in terms of cost but also in terms of time scheduling.

We studied the DIA model with time constraints based on a case study derived from the QO model in chapter 4. This QO model models the Quotation Optimization process of an airline component workshop that responds to Requests for Quotation. In this chapter we have added some extra information attributes to better represent reality and increase the complexity of the model. Moreover, we use the earlier defined acquisition times that also determine the acquisition cost to be able to performm optimal parallel information acquisition.

Since the model that we studied is complex and computationally too large to solve exactly, we propose a Deep Reinforcement Learning method that is introduced by van Jaarsveld [2020]. Besides adapting this algorithm to apply it in the best way suitable for the DIA model with time constraints, we impose some extra constraints that are derived specifically based on the assumptions of the DIA model. For a small base case we are able to compare the solution with optimal solution and show that the gap is very small.

For the QO model, due to the complexity, it is not possible to compare the DRL solution to the optimal solution. Therefore, we compare it to a practical benchmark policy. This policy does not collect any information but always optimizes immediately the final decision. We show a significant gap between the benchmark and the DRL policy.

Appendix

5.A. Variability Scenarios

Table 5.4: Variability scenarios, each row gives the probabilities for the different values of the attribute according to Table 5.1

Scenario 1	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
SK	0.06	0.12	0.18	0.24	0.22	0.12	0.06	
MaC	0.06	0.12	0.18	0.24	0.32	0.06	0.02	
MH	0.06	0.12	0.18	0.24	0.32	0.06	0.02	
Cutil	0.06	0.12	0.18	0.24	0.32	0.06	0.02	
EMC	0.25	0.25	0.25	0.25				
PMA	0.5	0.125	0.125	0.125	0.125			
Num	0.3	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Period	0.1	0.4	0.3	0.2				
μ	0.3	0.25	0.2	0.1	0.05	0.05	0.05	
Scenario 2								
SK	0.03	0.06	0.24	0.3	0.28	0.06	0.03	
MaC	0.03	0.06	0.24	0.3	0.28	0.06	0.03	
MH	0.03	0.06	0.24	0.3	0.28	0.06	0.03	
Cutil	0.03	0.06	0.24	0.3	0.28	0.06	0.03	
EMC	0.125	0.375	0.375	0.125				
PMA	0.8	0.05	0.05	0.05	0.05			
Num	0.6	0.1	0.05	0.05	0.05	0.05	0.05	0.05
Period	0.2	0.4	0.3	0.1				
μ	0.5	0.15	0.1	0.1	0.05	0.05	0.05	
Scenario 3								
SK	0.12	0.12	0.12	0.28	0.12	0.12	0.12	
MaC	0.12	0.12	0.12	0.28	0.12	0.12	0.12	
MH	0.12	0.12	0.12	0.28	0.12	0.12	0.12	
Cutil	0.12	0.12	0.12	0.28	0.12	0.12	0.12	
EMC	0.375	0.125	0.125	0.375				
PMA	0.25	0.25	0.25	0.125	0.125			
Num	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
Period	0.2	0.3	0.3	0.2				
μ	0.1	0.15	0.2	0.2	0.15	0.15	0.05	

6

Policies for the Dynamic Traveling Maintainer Problem with Alerts

6.1. Introduction

Industrial assets such as wind turbines, trains, medical imaging equipment, aircraft, and wafer steppers are required to experience minimal downtime. Ideally, assets are maintained just before failure so as to ensure the highest availability at minimum cost. However, the failure mechanism of such assets is *a priori* typically unknown.

To keep assets functioning as intended, various maintenance policies exist: (i) Assets are maintained at scheduled times, i.e., *time-based maintenance* (TBM) planning; or (ii) Assets are regularly inspected to evaluate their degradation, i.e., *condition-based maintenance* (CBM). CBM is increasingly prevalent following the rise of smart monitoring devices and predictive models. Indeed, predictive models can use real-time sensor readings to predict the future failure times of assets. These predictions are typically communicated to the central operation in the form of *alerts*, which serve as an early indication of potential failures [Topan et al., 2020]. Numerous models exist that optimize maintenance timing for individual assets. However, the ability to maintain individual assets is typically subject to the availability of maintenance resources that are shared over a network of geographically dispersed assets. In such

This chapter is based on Da Costa et al. [2023].

networks, predictive models yield lists of alerts and predicted failure moments that need to be prioritized. Since alerts often carry uncertain information about the failure time due to misreadings, sensor malfunction or prediction errors, devising effective policies for these cases entails considering the uncertainty of alerts in addition to maintenance costs and travel times.

Examples of asset networks that regularly require maintenance during their long operational life include offshore wind-turbine parks. Such parks can contain over 100 turbines and are often built in a grid-like structure, where pairwise distances between turbines are equal [Breton and Moe, 2009]. To maintain wind turbines, maintenance engineers circulate on a vessel carrying equipment for repairs. In this case, the use of remote monitoring devices and failure predictions can represent a significant improvement over TBM policies. For example, a vibration sensor connected to the gearbox of the turbine can be used to detect signs of degradation [Kenbeek et al., 2019]. Similar to wind-turbine parks, manufacturers of medical equipment (such as scanners) have to maintain their assets within a network of hospitals. Medical equipment is equipped with an array of sensors that track medical procedures, but this data can also be used to signal degradation, which in turn may trigger engineer dispatching.

We conceptualize (what we shall refer to from now onward as) the dynamic traveling maintainer problem with alerts (DTMPA) and study the optimization of maintenance decisions in the presence of alerts, under the combination of the following challenges: (i) limited repair capacity and travel times; and (ii) asset degradation that is stochastic and typically unknown. In the DTMPA, an engineer travels in a network, where each location contains a single asset. The degradation of the individual assets is modeled through: (i) A finite number of degradation states; and (ii) A set of three phases (healthy, alert and failed). For the latter case, the degradation states are clustered in three phases and an alert is emitted when there is a transition from the *healthy* phase to the *alert* phase. The *failed* phase, corresponds to the state in which the asset has degraded to the extent that it is malfunctioning/down. The latter case corresponds to the case of censored/partial information of the degradation process.

The DTMPA is a discrete time sequential decision process. Each unit of time, the engineer can either travel to a location, wait or start a maintenance job at its current location. Preventive maintenance restores an asset from the alert phase to the healthy phase, while corrective maintenance restores an asset from the failed phase to the healthy phase. During maintenance, the asset is down. The objective is to minimize the total expected discounted cost, which consists of the costs for (preventive and corrective) maintenance and for the machine downtime.

When a machine transitions into the alert phase, it is valuable to use all available information to predict its residual lifetime. However, the available information may vary. Accordingly, we distinguish various levels of real-time information. These *information levels* range from no further information to full information (being able to observe the degradation state and knowing the remaining lifetime distribution). Using the various information levels, the proposed DTMPA framework can be tailored to a range of use cases.

We develop and compare three solution approaches for the DTMPA. First, we introduce a class of greedy, reactive heuristics that rank actions/assets based on maintenance costs, travel times and predicted failure times. Second, we propose a heuristic that leverages the alert information to construct a deterministic approximation of the problem similar to the traveling maintainer problem (TMP) [Camci, 2014]. Third, we propose an algorithm using distributional deep reinforcement learning [Bellemare et al., 2017] trained in a simulated environment optimizing total expected discounted maintenance costs. By using these solutions to go from alerts to decisions, we bridge the gap from predictive analytics to business prescriptive analytics for various levels of real-time degradation information.

The main contributions of this chapter are detailed as follows:

- We conceptualize the dynamic traveling maintainer problem with alerts on dispatching an engineer in an asset network.
- We introduce on the DTMPA various information levels depending on the available information retrieved from alerts.
- We propose heuristics yielding competitive policies for their respective information level compared to the optimal policy found under complete information on the degradation process.
- We illustrate that reinforcement learning (RL) outperforms other proposed heuristics, requiring only the minimum level of information and observations from the DTMPA.

The remainder of the chapter is organized as follows. Section 6.2 reviews related literature. Sections 6.3 and 6.4 introduce in detail the DTMPA framework. Section 6.5 discusses the various solutions methods. In Section 6.6, the setup of the numerical experiments is presented. Section 6.7 discusses the results and the corresponding managerial insights. We conclude the chapter in Section 6.8.

6.2. Related Work

We review three streams of relevant literature: maintenance optimization, traveling maintainer problems and lastly deep reinforcement learning for maintenance optimization.

Maintenance optimization models can be single-asset or multi-asset [Olde Keizer et al., 2017, de Jonge and Scarf, 2020]. Multi-asset models extend the work of single-asset models by considering joint maintenance policies for assets that exhibit any of four types of dependencies: economic, structural, stochastic and resource dependency [Olde Keizer et al., 2017]. Asset degradation is typically modeled with a stochastic process defined on a discrete finite state-space, e.g., a Markov chain. Models with three degradation states can also be viewed as delay-time models: The second state may represent a *defect* phase, which can be determined by inspection [Wang, 2012]. One can improve scheduled inspections with information acquired via sensors. These sensors sometimes measure asset degradation directly, but typically they measure asset health indirectly [van Staden and Boute, 2021], for instance in the form of alerts [de Jonge et al., 2016, Akcay, 2022]. In a multi-asset scenario, practical problems include complex dependencies induced by the geographical layout, giving rise to the so-called traveling maintainer problems.

The objective of the classical *traveling maintainer problem* (TMP) is to find a path through a network of assets which minimizes the sum of the time needed to reach each asset. The TMP is obtained from the traveling salesman problem (TSP) as a mean-flow variant, and is thus NP-complete [Afrati et al., 1986]. One can also assign a hard deadline to each asset, such as a bound on the response time, which further increases complexity. Tulabandhula et al. [2011] apply machine learning to estimate time-dependent failure probabilities at asset locations and propose two TMP objectives; learn the failure probabilities and solve the TMP sequentially or simultaneously. Camci [2014] studies the TMP with the objective of minimizing the sum of functions of response times to assets. This variant, which integrates real-time CBM prognostics with the TSP by scheduling maintenance using predicted failure information, was later extended to include travel times [Camci, 2015]. The dynamic TMP (DTMP) considers service demands that arrive uniformly in some region according to a Poisson process [Bertsimas et al., 1989]. These service demands must be fulfilled by a single maintainer, such that the average response time is minimal. Drent et al. [2020] formulate a variant of the DTMP as a Markov decision process (MDP) and propose dynamic heuristics for both the dispatching and repositioning of a repair crew using real-time condition information. Havinga and de Jonge [2020] investigate a cyclic TMP, formulated as an MDP, combined with condition-based preventive maintenance. The proposed control-limit heuristic is compared with the optimal policy and with corrective maintenance heuristics. A challenge with MDP-based solution methods is that large asset networks induce unwieldy state spaces, rendering any exact method computationally intractable. Heuristics and function approximation methods such as deep reinforcement learning (DRL) may be used to overcome these limitations.

Deep reinforcement learning in maintenance. Recent DRL approaches to maintenance problems include learning opportunistic maintenance strategies on parallel machines, resulting in downtime and cost reduction compared to reactive and time-based strategies [Kuhnle et al., 2019]. Compare et al. [2018] employ DRL to replacement-part management subject to stochastic failures. Andriotis and Papakonstantinou [2021] and Liu et al. [2020] consider inspection and maintenance policies to minimize long-term risk and costs for deteriorating assets. Similar to the DTMPA setting, this problem includes multiple challenges, such as high state cardinality, partial-observability, long-term planning, environmental uncertainties and constraints. Bhattacharya et al. [2020] consider policy rollouts and value function approximation to learn policies for a partially observable Markov decision process (POMDP) modeling a robot visiting adjacent machines. However, unlike the DTMPA case, these works do not consider maintenance optimization in an asset network. Note that DRL can potentially outperform hand-designed heuristic policies by learning to use the relevant state information from data requiring little human intervention.

In this chapter we combine the multiple research streams in maintenance optimization into a new framework for sequential decision-making of maintenance on asset networks under uncertainty. We consider a problem that is resource-dependent, as a single engineer maintains the entire asset network. As in previous works, we assume a finite state space per asset and allow for non-Markovian transition times between subsequent states. Moreover, we introduce three observable degradation states per asset, representing a censored observation of the true, hidden state. We assume that alerts received from sensors are related to asset degradation and contain imperfect residual lifetime predictions from black-box prediction models.

We adopt the objective to minimize the total expected discounted cost incurred due to downtimes of the assets and (unnecessary) maintenance actions. Instead of a hard deadline, we assume that the asset-dependent costs increase if the decision-maker does not perform preventive maintenance before the deadline triggered by the asset's failure mechanism. We develop greedy heuristics based on traditional rankings of alerts and a novel traveling maintainer heuristic based on optimizing the visitation order of a constructed TMP instance considering the urgency of the alerts and the near-future costs. Given the successes of DRL in maintenance optimization, we propose a DRL algorithm based on distributional RL [Bellemare et al., 2017], i.e., a class of methods that approximates the distribution of long-term costs. We learn policies that consider both maintenance and traveling decisions in a stochastic environment where observations are censored and alerts act as early indicators of failures.

6.3. Dynamic traveling maintainer problem with alerts

The DTMPA is a discrete-time model that considers a single decision-maker (maintenance engineer) and a set of assets (machines), denoted by $\mathcal{M} = \{1, \dots, M\}$, each positioned at a unique location in the network. Each asset $m \in \mathcal{M}$ degrades randomly over time and requires maintenance to prevent and resolve failures. In general, the *underlying* degradation state of the asset cannot be observed directly, and the decision-maker must rely on *alerts* that indicate early signs of degradation. These alerts will be used to characterize the healthy, alerted and failed phases of assets (cf. Section 6.3.1). Each period, the decision-maker selects an action u: Possible actions include traveling to another location, idling/continue or initiating maintenance at the current location. The goal is to minimize the total expected discounted cost (cf. Section 6.4.6). Preventive maintenance (PM) can be carried out before a failure and it restores the state of the asset to as-good-as-new. Corrective maintenance (CM) can be carried out after a failure has occurred and it also restores the state to as-good-as-new. Typically, a higher cost is paid for CM compared to PM. The action of maintenance takes time during which the machine is assumed to be down: Failed/down machines incur downtime costs. Traveling in the network takes time proportional to the distance between machine locations. Let $\theta_{ij} \in \mathbb{N}$ denote the travel time between assets $i, j \in \mathcal{M}$. We further assume that each asset $m \in \mathcal{M}$ degrades independently according to the *underlying* degradation model: In each period $t \in \mathbb{N}_0$, the asset's condition is denoted by state $x_m(t) \in \mathcal{N}_m$. Here, $\mathcal{N}_m = \{1, \ldots, |\mathcal{N}_m|\}$, where state $x_m^{\mathbf{h}} = 1$ denotes an as-good-as-new asset and state $x_m^{\mathbf{f}} = |\mathcal{N}_m|$ denotes a failed asset. Without intervention by the decision-maker, the next state is a "worse" state, i.e., $x_m(t+1) \geq x_m(t)$. To transition from state $x_m \in \mathcal{N}_m \setminus \{x_m^{\mathbf{f}}\}$ to state $x_m + 1$, it takes a random (positive, integer and independent) amount of time, say $T_m^{x_m}$. When asset m reaches the failed state $x_m^{\mathbf{f}}$, it resides in this state until a corrective maintenance action is completed.

6.3.1 Observations

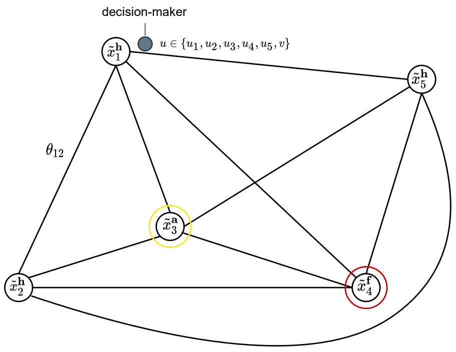

decision-maker
$u \in \{u_1, u_2, u_3, u_4, u_5, v\}$
$\tilde{x}_1^{\mathbf{h}}$
$\tilde{x}_5^{\mathbf{h}}$
θ_{12}
$\tilde{x}_3^{\mathbf{a}}$
$\tilde{x}_2^{\mathbf{h}}$
$\tilde{x}_4^{\mathbf{f}}$

Figure 6.1: Visualization of the DTMPA model for an asset network of $M = 5$ machines, plus the maintenance engineer. Based on observations, machines may be labeled as healthy, alerted (yellow) or failed (red). At discrete moments in time, an action u is selected; possible actions include idling/ continue (u_1), traveling to a location (u_2, u_3, u_4, u_5) and initiating maintenance at the current location (v).

Without intervention by the decision-maker, the random evolution of the degradation process evolves as follows: Initially, the state is set to $x_m^{\mathbf{h}}$ (as-good-as-new). After a random amount of time (required to pass through states $x_m^{\mathbf{h}}$ to $x_m^{\mathbf{a}}$), say $T_m^{\mathbf{a}}$, an alert is issued. After another (independent) random amount of time (required to pass through states $x_m^{\mathbf{a}}$ to $x_m^{\mathbf{f}}$), say $T_m^{\mathbf{f}}$ referred to as the *residual lifetime*, the asset fails.

The decision-maker typically cannot directly observe the evolution of the degradation state $x_m \in \mathcal{N}_m$ and can only observe the three phases: (healthy, alert and failed) captured in the set of observable phases, say $\mathcal{X}_m = \{\tilde{x}_m^{\mathbf{h}}, \tilde{x}_m^{\mathbf{a}}, \tilde{x}_m^{\mathbf{f}}\}$. The degradation states are mapped to these three phases as follows: If the degradation is in any of the states $\{x_m^{\mathbf{h}}, \ldots, x_m^{\mathbf{a}} - 1\}$, then the phase is healthy $(\tilde{x}_m^{\mathbf{h}})$. If the degradation is in any of the states $\{x_m^{\mathbf{a}}, \ldots, x_m^{\mathbf{f}} - 1\}$, then the phase is alert $(\tilde{x}_m^{\mathbf{a}})$. Lastly, the failed state corresponds to the failed phase $(\tilde{x}_m^{\mathbf{f}})$. Based on the observable transitions alone, the degradation model can be viewed as a delay time model. See Figure 6.1 for a visualization of an asset network.

6.3.2 Information levels

The decision-maker receives information through alerts, which enables a categorization of machines as healthy, alerted, or failed (cf. Section 6.3.1). When using this information for planning purposes, some approaches may need more information regarding the underlying degradation process than others. For example, information regarding the distribution of the residual lifetime $T_m^{\mathbf{f}}$ may enable more advanced planning approaches to better assess the risk of delaying maintenance on an alerted machine. We formalize this via four *levels of information*:

(**L**$_0$) The decision-maker acts on phase observations (cf. Section 6.3.1) and does not have any distributional information about the residual lifetime $T_m^{\mathbf{f}}$.

(**L**$_1$) The decision-maker acts on phase observations and has access to partial information (e.g., the expectation and the standard deviation) regarding the residual lifetime $T_m^{\mathbf{f}}$.

(**L**$_2$) The decision-maker acts on phase observations and has full information about the underlying model (the distribution of $T_m^{\mathbf{f}}$ is known).

(**L**$_3$) The decision-maker has full state information about the underlying model and observes all state transitions (at any point in time the exact state $x_m \in \mathcal{N}_m$ is known).

Given the information level, the objective is to produce a policy that minimizes the total expected discounted cost of a given asset network. The next section formalizes the DTMPA as a MDP.

6.4. Modeling framework

This section contains a formal description of the sequential decision-making DTMPA model framework. We introduce the underlying network states, actions, transitions, costs, observed information and the optimization objective.

6.4.1 Underlying/hidden network states

The *underlying/hidden network state* $h \in \mathcal{H}$ represents the state of all machines in the network as well as the status of the engineer. Each network state is a vector $h = (x_1, \ldots, x_M, \ell, \iota, \delta, \hat{t}_1, \ldots, \hat{t}_M) \in \mathcal{H}$. Here, $x_m \in \mathcal{N}_m$, $m \in \mathcal{M}$, denotes the degradation state of the m-th asset; $\ell \in \mathcal{M}$ denotes the location of the decision-maker; $\iota \in I = \{0, 1\}$ indicates whether the engineer is currently performing maintenance (as opposed to traveling or waiting); $\delta \in \Delta \subset \mathbb{N}$ denotes the remaining time units until the decision-maker becomes available again ($\delta = 0$ indicates that the decision-maker is idle); $\hat{t}_m$, $m \in \mathcal{M}$, denotes the *elapsed time*, i.e., the number of periods spent in the current state x_m. Inclusion of elapsed times in h ensures that the Markov properties are satisfied. Hence, $\mathcal{H} = \mathcal{N}_1 \times \ldots \times \mathcal{N}_M \times \mathcal{M} \times I \times \Delta \times \mathbb{N}^M$.

6.4.2 Actions

At every time instance, given $h = (x_1, \ldots, x_M, \ell, \iota, \delta, \hat{t}_1, \ldots, \hat{t}_M) \in \mathcal{H}$, the decision-maker can either: (i) Choose to start traveling immediately to another location, say u_m, $m \in \mathcal{M} \setminus \{\ell\}$; (ii) Choose to start a maintenance action at its current location, say v; or (iii) Continue with the current activity, including to remain idle, say u_ℓ, with ℓ denoting the current location. Note that when $\delta > 0$, the engineer is busy either performing maintenance or traveling and must therefore carry out action u_ℓ, while if $\delta = 0$, then the actions can be chosen from the set $\{u_m\}_{m \in \mathcal{M} \setminus \{\ell\}} \cup \{v\}$. The *state-dependent action set* thus becomes

$$\mathcal{U}(h) = \begin{cases} \{u_m\}_{m=1}^{M} \cup \{v\} & \text{if } \delta = 0, \\ \{u_\ell\} & \text{if } \delta > 0. \end{cases}$$

6.4.3 Transitions

The one-step state transitions $h \to h'$ are first determined by the chosen action, say $u \in \mathcal{U}(h)$, and then by the random evolution of the degradation processes. To this, $h \to h'$ is decomposed into $h \xrightarrow{u} h^u$ and to $h^u \xrightarrow{t \to t+1} h'$.

For $h = (x_1, \ldots, x_M, \ell, \iota, \delta, \hat{t}_1, \ldots, \hat{t}_M) \xrightarrow{u} h^u$,

$$h^u = \begin{cases} (x_1, \ldots, x_M, \ell, \iota, \delta, \hat{t}^u_1, \ldots, \hat{t}^u_M) & \text{if } u = u_m \wedge m = \ell, \\ (x_1, \ldots, x_M, m, 0, \theta_{\ell m}, \hat{t}^u_1, \ldots, \hat{t}^u_M) & \text{if } u = u_m \wedge m \neq \ell, \\ (x_1, \ldots, x_{\ell-1}, x^{\mathbf{f}}_\ell, x_{\ell+1}, \ldots, x_M, \ell, 1, t^{\mathrm{PM}}_\ell, \hat{t}^u_1, \ldots, \hat{t}^u_{\ell-1}, 0, \hat{t}^u_{\ell+1}, \ldots, \hat{t}^u_M) & \text{if } u = v \wedge x_\ell \neq x^{\mathbf{f}}_\ell, \\ (x_1, \ldots, x_M, \ell, 1, t^{\mathrm{CM}}_\ell, \hat{t}^u_1, \ldots, \hat{t}^u_M) & \text{if } u = v \wedge x_\ell = x^{\mathbf{f}}_\ell. \end{cases}$$

In the first case, the decision-maker stays at the current location $m = \ell$ and carries on with the ongoing action, including to remain idle, (u_ℓ). In the second case, the decision-maker initiates travel: The remaining unavailability δ changes to the travel time $\theta_{\ell m}$ and the location ℓ changes to the destination $m \in \mathcal{M} \setminus \{\ell\}$. The third and fourth case represent initiating maintenance at the current location, where the maintenance type (PM or CM) depends on the status x_ℓ of the machine at the location of the engineer. The duration of maintenance is either $t^{\mathrm{PM}}_m \in \mathbb{N}^+$ or $t^{\mathrm{CM}}_m \in \mathbb{N}^+$, and δ changes accordingly. Initiating maintenance on machine m advances the degradation state x_m to $x^{\mathbf{f}}_m$, such that the machine is unavailable during maintenance, in line with modeling assumptions.

For $h^u = (x^u_1, \ldots, x^u_M, \ell^u, \iota^u, \delta^u, \hat{t}^u_1, \ldots, \hat{t}^u_M) \xrightarrow{t \to t+1} h' = (x'_1, \ldots, x'_M, \ell', \iota', \delta', \hat{t}'_1, \ldots, \hat{t}'_M)$, the determination of h' is performed in two steps. In step 1, we check for every machine if there is further degradation or completion of maintenance. In step 2, we update the remaining state variables of the model. The state changes as follows:

- The machines evolve according to the following set of cases: If $(\ell^u, \iota^u, \delta^u) = (m, 1, 1)$, then $(x'_m, \hat{t}'_m) = (x^{\mathbf{h}}_m, 0)$. This first case represents completion of a maintenance job, resulting in a transition to the as-good-as-new state. Else, with probability $\mathbb{P}(\hat{t}^u_m) := \mathbb{P}(T^{x^u_m}_m = \hat{t}^u_m + 1 | T^{x^u_m}_m > \hat{t}^u_m)$, $(x'_m, \hat{t}'_m) = (x^u_m + 1, 0)$.

This second case represents the transition to a subsequent degradation state and $\mathbb{P}(\hat{t}^u_m)$ denotes the corresponding probability. Otherwise, with probability $1 - \mathbb{P}(\hat{t}^u_m)$, $(x'_m, \hat{t}'_m) = (x^u_m, \hat{t}^u_m + 1)$. This third case captures that the machine degradation state does not change, which implies that the elapsed time since the last transition increases by one.

- $(\ell^u, \max(\delta^u - 1, 0), \iota^u) \overset{t \to t+1}{\longrightarrow} (\ell', \delta', \iota')$; i.e., the remaining unavailability δ' of the engineer, due to an ongoing action, is decreased by one, if possible.

6.4.4 Cost structure

Initiating maintenance on machine $m \in \mathcal{M}$ incurs costs $c^{\mathrm{PM}}_m \in \mathbb{R}^+$ or $c^{\mathrm{CM}}_m \in \mathbb{R}^+$, for PM or CM, respectively. We assume that $c^{\mathrm{CM}}_m \geq c^{\mathrm{PM}}_m$. An asset m is said to be down when it is in the failed state or it is under repair, viz. in state $x^{\mathbf{f}}_m$. The cost of downtime is $c^{\mathrm{DT}}_m \in \mathbb{R}^+$ per time unit, independently of the cause of downtime. Summing up, when taking action $u \in \mathcal{U}(h)$ in state h, the incurred costs are:

$$C(h, u) = c^{\mathrm{CM}}_\ell \mathbb{1}_{\{u=v, x_\ell = x^{\mathbf{f}}_\ell\}} + c^{\mathrm{PM}}_\ell \mathbb{1}_{\{u=v, x_\ell < x^{\mathbf{f}}_\ell\}} + \sum_{m \in \mathcal{M}} c^{\mathrm{DT}}_m \mathbb{1}_{\{x_m = x^{\mathbf{f}}_m\}}. \tag{6.1}$$

6.4.5 Observed network states and censoring

For information levels $\mathbf{L}_0$ to $\mathbf{L}_3$, the underlying state $h = (x_1, \ldots, x_M, \ell, \iota, \delta, \hat{t}_1, \ldots, \hat{t}_M)$ is not fully observable. Instead, one observes $o = (\tilde{x}_1, \ldots, \tilde{x}_M, \ell, \iota, \delta, \tilde{t}_1, \ldots, \tilde{t}_M) \in \Omega = \mathcal{X}_1 \times \ldots \times \mathcal{X}_M \times \mathcal{M} \times I \times \Delta \times \mathbb{N}^M$, where, for $m \in \mathcal{M}$, $\tilde{x}_m \in \mathcal{X}_m$, cf. Section 6.3.1, and where $\tilde{t}_m$ corresponds to the time since machine m transitioned to observable phase $\tilde{x}_m$.

6.4.6 Objective

We are interested in a policy, say $\pi^{\mathbf{L}}$, with $\mathbf{L} \in \{\mathbf{L}_0, \mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3\}$ denoting the information level, which minimizes the total expected discounted cost. More precisely, a policy is defined as a sequence of decision rules, i.e., $\pi^{\mathbf{L}} = (\pi^{\mathbf{L}}_1, \pi^{\mathbf{L}}_2, \ldots, \pi^{\mathbf{L}}_t, \ldots)$, where the decision rule $\pi^{\mathbf{L}}_t$ assigns the probability of the action to be taken at time t, given the history (observed states from time 0 until time t and the respective actions).

Let $\gamma \in [0, 1)$ be the discount factor and $J(\pi^{\mathbf{L}})$ be the total expected discounted cost. Then, the objective is to find the optimal policy $\pi^{\mathbf{L}}_*$ satisfying

$$\pi^{\mathbf{L}}_* = \arg\min_{\pi^{\mathbf{L}}} J(\pi^{\mathbf{L}}) = \arg\min_{\pi^{\mathbf{L}}} \lim_{T\to\infty} \mathbb{E}_{\pi^{\mathbf{L}}} \left[\sum_{t=0}^{T} \gamma^t C\left(O(t), U(t)\right) \middle| O(0) = o \right], \quad (6.2)$$

where $(O(t), U(t))$ denotes the tuple of the observed state and the respective action given the policy $\pi^{\mathbf{L}}$ at time t, $t \geq 0$ and $C(\cdot)$ denotes the associated cost (maintenance and downtime).

6.5. Solution approaches

In this section, we discuss three solution approaches for the introduced class of problems.

6.5.1 Ranking heuristics

In this section, we construct a simple class of $\mathbf{L}_1$ heuristics: The *reactive* heuristic, which maintains only failed machines, and the *greedy* heuristic, which maintains both alerted and failed machines. To select which machine to maintain next, both heuristics require the introduction of a *ranking* between assets. The ranking is made on the premise of failures and on the premise of alerts, accounting also for travel times and costs.

Failure time (F). Given the alert information, the machines are ranked on their estimated failure time. Assets in the failed state are prioritized over any of these potential future failures. Let $t \in \mathbb{N}_0$ be the current time and $t^{\mathbf{a}}_m$ be the time at which the last alert for the machine was generated, and define the failure time estimate $\tilde{T}^{\mathbf{f}}_m = \max\left\{t, t^{\mathbf{a}}_m + \mathbb{E}\left[T^{\mathbf{f}}_m\right]\right\}$.

Travel time (T). Assets are ranked based on proximity to the engineer. For distant assets, the travel times will be long, meaning valuable time will be lost whilst no maintenance is conducted. The decision-maker opts for maintaining assets over traveling.

Cost savings (C). To account for assets with different cost structures (economic risk), we adopt a ranking that is constructed as follows: For assets in the alert state, suppose asset m, we compute the immediate benefit of PM over CM, specifically $c_m = (c_m^{\mathrm{CM}} - c_m^{\mathrm{PM}}) + (t_m^{\mathrm{CM}} - t_m^{\mathrm{PM}})c_m^{\mathrm{DT}}$. For assets in the failed state, suppose m', we compute the total downtime cost when deciding to directly repair the asset, given by $c_{m'} = (\theta_{\ell,m'} + t_{m'}^{\mathrm{CM}})c_{m'}^{\mathrm{DT}}$. The aim of this ranking is to prioritize the machines which carry the highest economic risk.

For both heuristics, the assets to be maintained are ranked based on the three rankings in a chosen order, e.g., [F,T,C]. The reactive heuristic (in contrast to the greedy heuristic) is obtained by ranking only the failed assets. Heuristics like the ones presented above are a reasonable choice for companies faced with problems similar to the DTMPA; while attention is restricted to two variants here, a rich range of ranking heuristics could be further identified.

6.5.2 Traveling maintainer heuristic

The *traveling maintainer heuristic* (TMH) policy is inspired by the traveling maintainer problem (TMP) that arises for a group of assets which require maintenance at deterministic maintenance deadlines. Namely, the TMH policy globally consists of the following two steps:

First order approximation. In this step, we approximate the random times that an alerted asset might fail by deterministic maintenance deadlines. Using these approximations, the problem reduces to a deterministic TMP.

Optimization step. Solve the constructed TMP instance by optimizing the *schedule*, i.e., the time at which maintenance is initiated for all assets, for each possible order in which the assets can be visited. A schedule is optimal when it achieves the minimum discounted cost in the deterministic optimization problem; ties are broken by selecting uniformly at random from the schedules with the longest duration.

When applying the THM policy at time $t \in \mathbb{N}_0$, we first construct a *first order approximation*. We compute deterministic maintenance deadlines, say t_m, for the various failed and alerted assets. For failed assets, we set t_m equal to the corresponding failure time $t_m^{\mathbf{f}} \leq t$. For alerted assets, it may be advantageous to postpone maintenance, as maintenance incurs costs and induces costly downtime. Accordingly, for alerted assets, we compute t_m as the cost-optimal maintenance time τ associated with a time-based maintenance (TBM) policy. Note that effective DTMPA downtime costs depend on the *response time* (the time needed before the engineer starts maintenance at the asset location), which is 0 when $|\mathcal{M}| = 1$, but which is difficult to characterize when $|\mathcal{M}| > 1$. Assuming a response time of 0 results in a τ-TBM policy with total expected discounted cost given by:

$$J(\tau) = \frac{\bar{c}_m^{\mathrm{CM}}\mathbb{E}[\gamma^{T_m^{\mathbf{a}}+T_m^{\mathbf{f}}}\mathbb{1}_{\{T_m^{\mathbf{f}}\leq\tau\}} \mid X_m(0) = x_m^{\mathbf{h}}] + \bar{c}_m^{\mathrm{PM}}\mathbb{E}[\gamma^{T_m^{\mathbf{a}}+\tau}\mathbb{1}_{\{T_m^{\mathbf{f}}>\tau\}} \mid X_m(0) = x_m^{\mathbf{h}}]}{1 - \gamma^{t_m^{\mathrm{CM}}}\mathbb{E}[\gamma^{T_m^{\mathbf{a}}+T_m^{\mathbf{f}}}\mathbb{1}_{\{T_m^{\mathbf{f}}\leq\tau\}} \mid X_m(0) = x_m^{\mathbf{h}}] - \gamma^{t_m^{\mathrm{PM}}}\mathbb{E}[\gamma^{T_m^{\mathbf{a}}+\tau}\mathbb{1}_{\{T_m^{\mathbf{f}}>\tau\}} \mid X_m(0) = x_m^{\mathbf{h}}]}. \tag{6.3}$$

A proof of this claim can be found in 6.A. The optimal τ-TBM policy[1] τ^* can be found by numerically computing $\tau^* = \arg\min J(\tau)$. Now, for an alerted machine, say m, with corresponding alert time t_a, we set the deadline as:

$$t_m = \max\{t, t_a + \tau^* + 1\}. \tag{6.4}$$

It is now assumed that asset m fails at time t_m, thus maintenance is preferably initiated at time $t_m - 1$. Accordingly, the *optimization step* plans a cost-efficient route and schedule to maintain the failed and alerted assets.

Let $\mathcal{M}_t$ denote the subset of all failed and alerted machines at time t and let $\Sigma_{\mathcal{M}_t}$ denote the permutations of $\mathcal{M}_t$. For each path $\sigma \in \Sigma_{\mathcal{M}_t}$, we construct an initial schedule (cf. Algorithm 1 in 6.B) that is tight in the sense that travel actions and maintenance actions are executed without intermediate delay. To improve this schedule, we may introduce intermediate delay/slack. To simplify slack assignment, we identify conditions such that any voluntary violation of maintenance deadlines is sub-optimal in the constructed TMP. Define the combined costs $\bar{c}_m^{\mathrm{CM}} = c_m^{\mathrm{CM}} + c_m^{\mathrm{DT}}\frac{\gamma^{t_m^{\mathrm{CM}}}-1}{\gamma-1}$ and $\bar{c}_m^{\mathrm{PM}} = c_m^{\mathrm{PM}} + c_m^{\mathrm{DT}}\frac{\gamma^{t_m^{\mathrm{PM}}}-1}{\gamma-1}$, where $\sum_{i=1}^{t}\gamma^{i-1} = \frac{\gamma^t-1}{\gamma-1}$. Voluntary violations of maintenance deadlines are suboptimal if the following four conditions hold for all m: (i) $\gamma\bar{c}_m^{\mathrm{CM}} > \bar{c}_m^{\mathrm{PM}}$ (it is loss-making to perform CM instead of PM), (ii) $c_m^{\mathrm{DT}} + \gamma\bar{c}_m^{\mathrm{CM}} > \bar{c}_m^{\mathrm{CM}}$ (it is loss-making to postpone a CM action), (iii) $\gamma\bar{c}_m^{\mathrm{CM}} - \bar{c}_m^{\mathrm{PM}} > \sum_{m'\in\mathcal{M}\setminus\{m\}}(1-\gamma)\bar{c}_{m'}^{\mathrm{PM}}$ (it is loss-making to perform CM instead of PM to delay an arbitrary number of other PM actions) and (iv) $c_m^{\mathrm{DT}} + (\gamma-1)\bar{c}_m^{\mathrm{CM}} > \sum_{m'\in\mathcal{M}\setminus\{m\}}(1-\gamma)\bar{c}_{m'}^{\mathrm{PM}}$ (it is loss-making to delay a CM action to delay an arbitrary number of PM actions).

To find an optimal schedule for a given route, we thus first delay the last repair as

[1]The *average cost criterion* can be computed by evaluating the limit of $(1-\gamma)J(\tau)$ as $\gamma \to 1$.

much as possible, which will allow us to delay the previous repair as well, and so on. See Algorithm 2 in 6.B for details. Given a path σ and the optimized schedule $\mathbf{t}^{\sigma}_{\text{opt}}$, the tuple $(\sigma, \mathbf{t}^{\sigma}_{\text{opt}})$ induces a policy in the DTMPA, which induces a trajectory of observations $o_t, \ldots, o_{\mathbf{t}^{\sigma}(2|\mathcal{M}_t|+1)}$ for which we can compute the discounted cost. Select the tuple $(\sigma, \mathbf{t}^{\sigma}_{\text{opt}})$ that minimizes the discounted costs.

The heuristic advantages are numerous. Firstly, the heuristic considers jointly the network layout, cost structures and the information captured in the alerts. Secondly, one only needs to solve a TMP when a new alert or failure arrives, otherwise the previous solution can continue to be used for planning actions. A disadvantage concerns the algorithm complexity as $|\Sigma_{\mathcal{M}_t}| = |\mathcal{M}_t|!$, which in the worst case implies a run time of $\mathcal{O}(2^M)$.

The TMH policy picks the path where it meets the most costly maintenance deadlines, preferably scheduling maintenance at time $t_m - 1$ but possibly expediting this decision to ensure that as many maintenance deadlines as possible are met. The policy can adequately assess the immediate benefit of PM over CM, but it may potentially be less suitable when response times are substantial and downtime costs are the dominant factor.

6.5.3 Deep reinforcement learning

We adopt Eq.(6.2) as a DRL objective under $\mathbf{L}_0$. In the corresponding POMDP formulation (cf. Section 6.4.5), observable network states summarize the relevant *history* in the form of elapsed times since the last observable transition. DRL is applicable when the proposed environment can be simulated for a number of episodes $E \in \mathbb{N}$. Note that this is possible when learning online by observing past state transitions. We could also reuse transitions from previous policies or implement a simulation environment that mimics real-world data before deploying the learned agents.

We seek to learn a policy $\pi^{\mathbf{L}_0}$, mapping observable network states to action probabilities. Learning is possible by sampling a number of trajectories following a policy, and evaluating its performance with respect to the main objective in Eq.(6.2). To evaluate performance, we define the standard characterization of the state-action value function measuring the expected sum of costs starting from a given observable network state o, action u, starting at time t as

$$q^{\pi^{\mathbf{L}_0}}(o,u) = \mathbb{E}_{\pi^{\mathbf{L}_0}}\left[\sum_{k=0}^{\infty} \gamma^k C\left(O(t+k), U(t+k)\right) | O(t) = o, U(t) = u\right]. \tag{6.5}$$

To avoid notation clutter, we refer to the class of DRL policies interchangeably as $\pi^{\mathbf{L}_0}$ and π, costs at time t as $c_t := C\left(O(t), U(t)\right)$ and observable network states as $o_t := O(t)$. We assume no access to the environment transition probabilities (model), in which case a policy π may be learned as a mapping from states to actions or extracted from value function approximations. We propose a value (distribution) approximation method, referred to as n-step quantile regression double Q-Learning (nQR-DDQN) detailed in the forthcoming sections.

n-step quantile regression double Q-Learning

Similar to [Hessel et al., 2018, Badia et al., 2020], we extend deep Q-Learning (DQN)[Mnih et al., 2015] by combining several modifications that improve performance, training times and maintain comparable sampling complexity [Hessel et al., 2018, Dabney et al., 2018, Van Hasselt et al., 2016], detailed below. In DQN, off-policy RL is combined with neural network (NN) approximation to estimate values of state-action pairs. State information is passed to a NN $q_w : \Omega \times \mathcal{U} \to \mathbb{R}$, where w are trainable parameters. Similar to the original DQN, we store a set of transitions (o_t, u_t, c_t, o_{t+1}) in a *replay memory* $\mathcal{D}$ (hash table). Transitions are collected following an ϵ-greedy exploration scheme, i.e., picking a random action with probability ϵ, otherwise selecting the action with the lowest estimated q values. For training, we sample a mini-batch of $N_B \in \mathbb{N}$ transitions from $\mathcal{D}$ uniformly at random to decorrelate past transitions. Based on the mean Bellman optimality operator, the parameters of the NN are trained to minimize the one-step Temporal Difference (TD[0]) error [Sutton and Barto, 2018, p. 131], i.e., the error between the current estimate of the value of a state and its one-step update, acting greedily with respect to the approximated q function via a mean squared error (MSE) loss function. DQN introduces a *target network* with parameters $\bar{w}$, i.e., a periodic copy of the online q_w that is not directly optimized and used as a target for the future q-value estimates. This copy stabilizes training and represents a fixed target that does not change at each update of w [Mnih et al., 2015]. This copy is updated every $P \in \mathbb{N}$ episodes.

Double deep Q-Learning DQN can be affected by an underestimation bias (overestimation in maximization) [Van Hasselt et al., 2016]. In a noisy environment, this can lead to over-optimistic actions and prevent learning. Thus, we employ double DQN (DDQN) [Van Hasselt et al., 2016] to decouple the action selection from its evaluation. In this way, the action selection when updating w remains dependent on the online q_w (the current network being optimized), effectively reducing the chance of underestimating action values on the target network $q_{\bar{w}}$. DDQN effectively changes the loss of DQN to

$$\mathcal{L}(w) = (c_t + \gamma q_{\bar{w}}(o_{t+1}, \arg\min_{u'} q_w(o_t, u')) - q_w(o_t, u_t))^2. \tag{6.6}$$

Multi-step reinforcement learning One-step temporal difference updates of DQN are biased when q_w estimates are far from true q^π values. To obtain better estimates, TD[0] updates can be generalized by bootstrapping over longer horizons. This reduces bias but it can lead to high variance due to the environment's stochasticity [Jaakkola et al., 1994]. Nevertheless, using a larger bootstrapping horizon can empirically lead to higher performance and faster learning. We include longer horizons considering the truncated n-step future discounted costs

$$C^{t:t+n} = c_t + \gamma c_{t+1} + \gamma^2 c_{t+2} + \ldots + \gamma^{n-1} c_{t+n-1}. \tag{6.7}$$

Using such costs makes the updates of DQN on-policy and results in value function updates that rely on old and inaccurate transitions in the replay memory. When the policy that generates the transitions in the replay memory differs substantially from the current policy, the resulting updates can cause inaccurate evaluations of the q function under the current policy. To make updates off-policy again, we need to introduce importance sampling terms [De Asis et al., 2018]. However, these terms can also lead to higher variance. In practice, adding n-step terms (even without importance sampling) can aid learning, whilst high variance can be controlled by small values of n [Hernandez-Garcia and Sutton, 2019]. Thus, to balance bias (one-step updates) and variance (multi-step updates), we include an additive n-step term weighted by $\alpha \in \mathbb{R}$ in the original DQN training objective. The parameter α is introduced to control how much extra variance we allow when updating the estimated q values.

Distributional reinforcement learning We consider the failure of crucial assets, in which risk-adjusted costs need to be considered when selecting actions, thus learning a distribution of future costs allows us to improve policy performance by taking into account the stochasticity of the problem explicitly. Distributional RL [Bellemare et al., 2017] aims to learn not a single point estimate of values but a distribution of returns, i.e., the distribution of the random variable $z^{\pi}(o,u) = \sum_{k=0}^{\infty} \gamma^k c_{t+k}$ for given o and u, where $q^{\pi}(o,u) := \mathbb{E}_{\pi}[z^{\pi}(o,u)]$ [Bellemare et al., 2017]. We employ quantile regression DQN (QR-DQN) [Dabney et al., 2018], in which the distribution of returns is modeled via a quantile regression on N data points with fixed uniform weights of the CDF of z^{π} as $\tau_i = \frac{i}{N}$, $i = 1, \ldots, N$.

Thus, we estimate the quantiles by learning a model $\psi_w : \Omega \times \mathcal{U} \to \mathbb{R}^N$, mapping each state-action pair to a probability distribution supported on $\{\psi_w^i(o,u)\}_{i=1}^N$, i.e., $z_{\psi}(o,u) := \frac{1}{N}\sum_{i=1}^N \delta_{\psi_w^i(o,u)}$ where δ_z represents a Dirac at $z \in \mathbb{R}$. Moreover, each ψ_w^i represents an estimation of the quantile value corresponding to the CDF quantile weight $\hat{\tau}_i = \frac{\tau_{i-1}+\tau_i}{2}$, $\tau_0 = 0$, i.e., the data points that minimize the 1-Wasserstein metric to the true z^{π} [Dabney et al., 2018]. Note that $z_{\psi}(o,u)$ can be used to compute the usual q_w estimates as $q_w(o,u) = \frac{1}{N}\sum_{i=1}^N \psi_w^i(o,u)$. To achieve unbiased gradients in QR-DQN, we replace the usual MSE loss of DQN with an asymmetric variant of the Huber loss [Huber, 1964] defined as $\rho_{\hat{\tau}_i}^{\kappa}(\nu) = |\hat{\tau}_i - \mathbb{1}_{\{\nu<0\}}|\mathcal{L}_{\kappa}(\nu)$, where

$$\mathcal{L}_{\kappa}(\nu) = \begin{cases} \frac{1}{2}\nu^2 & \text{for } |\nu| \leq \kappa, \\ \kappa(|\nu| - \frac{1}{2}\kappa), & \text{otherwise,} \end{cases} \tag{6.8}$$

ν corresponds to pairwise TD errors and $\kappa \in \mathbb{R}$. Similar to the mean Bellman optimality operator, the distributional Bellman optimality operator [Bellemare et al., 2017], i.e., $z^{\pi}(o,u) = C_u(o) + \gamma z^{\pi}(o', \arg\min_{u'} \mathbb{E}_{\pi}[z^{\pi}(o',u')])$, where o' is the next observable network state, is used to approximate quantiles based on the n-step loss function (with DDQN correction) as:

$$\mathcal{L}_{QR}^{t:t+n}(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{i'=1}^{N} \rho_{\hat{\tau}_i}^{\kappa}(y_{t:t+n}^{i'} - \psi_w^i(o_t,u_t)), \tag{6.9}$$

where $y_t^{i'} = C^{t:t+n} + \gamma^n \psi_{\bar{w}}^{i'}(o_{t+n}, \arg\min_{u'} \sum_{i=1}^N \psi_w^i(o_{t+n},u'))$. Finally, we add the

n-step QR loss term to a single-step term to arrive at the objective:

$$w^* = \arg\min_w \hat{\mathcal{L}}_{n\text{QR-DDQN}}(w) = \arg\min_w \left[\mathcal{L}_{QR}(w) + \alpha\mathcal{L}_{QR}^{t:t+n}(w)\right], \tag{6.10}$$

note that $\mathcal{L}_{QR}(w)$ is the same as $\mathcal{L}_{QR}^{t:t+1}(w)$ and corresponds to TD[0] (one-step) updates. This is the training objective for nQR-DDQN with the complete algorithm depicted in 6.C.

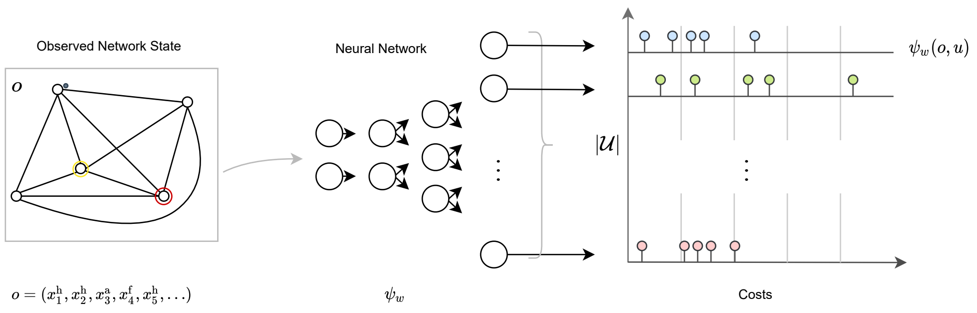

Observed Network State
o
$o = (x_1^{\text{h}}, x_2^{\text{h}}, x_3^{\text{a}}, x_4^{\text{f}}, x_5^{\text{h}}, \dots)$
Neural Network
ψ_w
$|\mathcal{U}|$
$\psi_w(o,u)$
Costs

Figure 6.2: A neural network $\psi_w : \mathbb{R}^d \to \mathbb{R}^{N\times|\mathcal{U}|}$, mapping observable state vectors to estimates of the quantile locations of the cost distributions.

Neural network architecture We employ a NN ψ_w, depicted in Figure 6.2, that maps raw observable network state vectors to pairs of actions and N quantiles of the distribution of z^π, i.e., $\psi_w : \mathbb{R}^d \to \mathbb{R}^{N\times|\mathcal{U}|}$. First, an observable network state is flattened into an input vector $\boldsymbol{\zeta} \in \mathbb{R}^d$, where $d = 3M + 2$. The vector $\boldsymbol{\zeta}$ is then passed through a series of layers $l \in \{1, \dots, L\}$ of the form $\boldsymbol{\zeta}^l = \sigma^l(\mathbf{W}^l\boldsymbol{\zeta}^{l-1} + \boldsymbol{b}^l)$, where $\mathbf{W}^l \in \mathbb{R}^{d^l\times d^{l-1}}$, $\boldsymbol{b}^l \in \mathbb{R}^{d^l}$, $d^l \in \mathbb{N}$, $d^0 = d$, $\boldsymbol{\zeta}^0 = \boldsymbol{\zeta}$ and σ^l is a (non-)linear activation function in every hidden layer, i.e., $l \in \{1, \dots, L-1\}$. In the output layer L, $d^L = |\mathcal{U}| \times N$, where $|\mathcal{U}|$ is the size of the action space and σ^L is the identity function. The parameters $w = \{\mathbf{W}^l, \boldsymbol{b}^l\}_{l=1}^L$ are updated via the gradient descent method, Adaptive Moment Estimation (Adam) [Kingma and Ba, 2015], computing adaptive learning rates for each parameter, with learning rate $\lambda > 0$ aimed at minimizing the loss in Eq.(6.10). Note that λ controls the magnitude of the gradient updates performed in Adam.

6.6. Experimental settings

To benchmark the algorithms discussed in Section 6.5, we construct several asset networks varying the number of machines $M \in \{1, 2, 4, 6\}$ in a network, while reducing complexity by assuming that $t_m^{\mathrm{PM}} = t_m^{\mathrm{CM}} = \theta_{mm'} \equiv 1, \forall m \neq m' \in \mathcal{M}$ and that machine degradation times are geometrically distributed (see Section 6.6.1). Under information level $\mathbf{L}_3$, the DTMPA then reduces to a rather manageable MDP: We can solve problems of up to four machines to optimality using policy iteration. With this exact benchmark in hand, we will be better positioned to assess the performance of the approaches developed in Section 6.5. Note that those approaches operate on information levels $\mathbf{L}_0 - \mathbf{L}_2$, for which the DTMPA becomes a POMPD with infinite state space which resists exact analysis. A detailed setup of the experiments follows in the remainder of this section. Experimental results are discussed in the following section.

6.6.1 Machine degradation

Machine degradation models in the experimental setup share the following characteristics. The alert state is the first non-healthy state, i.e., $x_m^{\mathbf{a}} \equiv x_m^{\mathbf{h}} + 1 = 2$. Transition times T_m^i between states $i, i+1 \in \mathcal{N}_m$ are geometrically distributed, i.e., $T_m^i \sim \mathrm{Geo}(p_m^i)$, where $p_m^i \in (0, 1)$ for all i and m. We assume $p_m^i = p_m^{\mathbf{a}}$ for all $i \in \{x_m^{\mathbf{a}}, \ldots, x_m^{\mathbf{f}} - 1\}$, thus $T_m^{\mathbf{f}} = \sum_{k=2}^{x_m^{\mathbf{f}}} \mathrm{Geo}(p_m^{\mathbf{a}}) \overset{\mathrm{d}}{=} x_m^{\mathbf{f}} - 2 + \mathrm{NegBin}(x_m^{\mathbf{f}} - 2, p_m^{\mathbf{a}})$. Geometrically distributed transition times enable us to capture degradation as a simple matrix Q (cf. Derman [1963]), where $Q_{i,i}$ denotes the per-period probability to remain in degradation state i, while $Q_{i,i+1}$ denotes the probability to transition to state $i+1$. Adopting this notation, we consider four types of machine degradation, denoted as follows:

$$\mathrm{Q1} = \begin{bmatrix} 0.8 & 0.2 & 0 \\ 0 & 0.7 & 0.3 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathrm{Q2} = \begin{bmatrix} 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.7 & 0.3 & 0 \\ 0 & 0 & 0 & 0.7 & 0.3 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Q3 = \begin{bmatrix} 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0.3 & 0.7 & 0 & 0 \\ 0 & 0 & 0.3 & 0.7 & 0 \\ 0 & 0 & 0 & 0.3 & 0.7 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad Q4 = \begin{bmatrix} 0.8 & 0.2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Degradation model Q1 has only three states, and hence $\mathcal{N} = \mathcal{X}$ rendering the model fully-observable. For machines that degrade following $Q1$, approaches developed in Section 6.5 could in theory achieve the performance of the optimal policy computed using policy iteration (PI) under information level $\mathbf{L}_3$. That is, the optimal solution to the MDP is in the class of $\mathbf{L}_0$ policies. This is no longer true if machines that degrade following matrices $Q2 - Q4$ are present. Degradation models Q2 and Q3 transition to an alert state with probability 0.2, but Q2 represents a slower degradation compared to Q3. Machine Q4 has more underlying states, while degradation is comparable to Q2.

6.6.2 Cost structure

We define three cost structures, namely C1, C2 and C3, which are presented in Table 6.1. Each cost structure represents a distinct, realistic cost relationship between preventive and corrective costs that induces unique optimal policies favouring more or less frequent maintenance actions. For example, when $\bar{c}^{\text{CM}}/\bar{c}^{\text{PM}}$ is large, i.e., when

Table 6.1: Cost structures considered in the experiments.

Cost Structure	c^{PM}	c^{CM}	c^{DT}	$c^{\text{CM}}+c^{\text{DT}}/c^{\text{PM}}+c^{\text{DT}}$
C1	0	9	1	10
C2	1	2	10	1
C3	1	4	1	2.5

CM costs greatly surpass PM costs, we expect that preventive maintenance policies outperform reactive policies.

6.6.3 Asset networks

We introduce 16 *asset networks*, each having a different combination of network size, cost structure and machine degradation matrices. We create various asset configurations for which a shorthand notation is adopted: First the number of assets is given, followed by the degradation matrices which are assumed to be spread evenly across the machines. For example, M4-Q2Q3 has four machines: Two with degradation matrix Q2 and two with matrix Q3.

We briefly discuss specifics regarding the instances. The simplest setup is M1-Q1; for this instance $\mathcal{X}_m = \mathcal{N}_m$ such that $\mathbf{L}_3$ has no informational advantage compared to $\mathbf{L}_0$. Network M1-Q4 considers a machine (Q4) where $\mathcal{X}_m \subsetneq \mathcal{N}_m$, resulting in informational advantage of $\mathbf{L}_3$ over the other information levels. To scale up the model complexity, we consider M2-Q2Q3 which considers two machines with different degradation rate, and M4-Q2Q3 which is obtained by doubling the M2-Q2Q3 network. The most challenging problems are obtained for M6-Q2Q3Q4: Apart from containing the most assets, note that the time between alert and failure is relatively long for Q4 machines, inducing a time-based maintenance problem where it is essential to adequately consider the risk of postponing maintenance, especially when maintenance resources are limited as is the case in the DTMPA.

These five asset configurations are considered with cost structures C1, C2 and C3. We obtain a final configuration M6-Q2Q3Q4-C by considering a custom cost structure on M6-Q2Q3Q4: Q2 machines are assigned cost structure C2, Q3 machines the cost structure C3 and Q4 machines have cost structure C1. In total, this gives rise to $5 \times 3 + 1 = 16$ DTMPA instances.

6.6.4 Reinforcement learning training parameters

For all experiments, we simulate for a maximum of $T = 500$ time steps[2] and $E = 2000$ episodes. Each new simulation episode is started with different alert/failure arrivals following identical probability distributions. We employ a NN comprised of a linear embedding and two layers with non-linear activations. All models are trained on a Ryzen 3950X CPU and a RTX 2080Ti GPU. The remaining parameters

[2]Assuming a maximum error level $\xi(1-\gamma)/c \approx 0.006$ between the infinite horizon costs and truncated horizon costs at time T, i.e., $T > \log(\xi(1-\gamma)c^{-1})/\log(\gamma) - 1$ where $c = 10$.

settings are presented in 6.D.

6.7. Experimental results

We generate 512 test episodes for each experimental setting. In the experiments, if the same deterministic policy is evaluated twice on an episode, the performance will be identical. We measure the solutions' performance and present the estimated total

Table 6.2: Average costs (95%CI) over 512 episodes and 500 steps. **Bold:** Lowest cost of the proposed approaches.

Solution (Info. Level)	Asset Network	Cost Structure		
		C1 [95% CI]	C2 [95% CI]	C3 [95% CI]
PI ($\mathbf{L}_3$)	M1-Q1	16.36	123.91	32.72
	M1-Q4	4.730	47.582	9.461
	M2-Q2Q3	21.230	190.275	39.550
	M4-Q2Q3	79.976	432.440	96.166
*n*QR-DDQN ($\mathbf{L}_0$)	M1-Q1	**16.365** [16.171, 16.56]	**124.96** [123.541, 126.378]	**32.804** [32.443, 33.165]
	M1-Q4	**8.806** [8.608, 9.004]	**47.408** [46.894, 47.922]	**14.513** [14.343, 14.683]
	M2-Q2Q3	**25.139** [24.935, 25.343]	**202.311** [200.675, 203.946]	**46.995** [46.609, 47.381]
	M4-Q2Q3	**92.654** [90.736, 94.571]	**470.625** [467.640, 473.611]	**106.525** [105.717, 107.334]
	M6-Q2Q3Q4	**176.642** [175.234, 178.051]	**711.188** [705.789, 716.586]	**159.527** [158.294, 160.760]
	M6-Q2Q3Q4-C		**347.500** [344.986, 350.014]	
H - Greedy [F,T,C] ($\mathbf{L}_1$)	M1-Q1	**16.365** [16.171, 16.56]	182.233 [180.126, 184.339]	**32.804** [32.443, 33.165]
	M1-Q4	16.61 [16.417, 16.804]	179.75 [177.624, 181.876]	32.818 [32.474, 33.163]
	M2-Q2Q3	30.9 [30.495, 31.305]	306.366 [304.26, 308.472]	56.692 [56.279, 57.105]
	M4-Q2Q3	112.304 [110.395, 114.212]	526.248 [523.62, 528.877]	112.306 [111.444, 113.168]
	M6-Q2Q3Q4	231.498 [228.491, 234.505]	741.568 [735.639, 747.497]	168.064 [166.677, 169.451]
	M6-Q2Q3Q4-C		379.799 [375.934, 383.665]	
H - Reactive [F,T,C] ($\mathbf{L}_1$)	M1-Q1	103.361 [102.217, 104.506]	**124.96** [123.541, 126.378]	51.736 [51.195, 52.278]
	M1-Q4	40.018 [39.595, 40.442]	**47.408** [46.894, 47.922]	20.127 [19.912, 20.342]
	M2-Q2Q3	154.074 [153.033, 155.114]	283.619 [281.469, 285.768]	82.419 [81.845, 82.993]
	M4-Q2Q3	306.278 [304.876, 307.68]	718.158 [713.699, 722.617]	173.682 [172.799, 174.565]
	M6-Q2Q3Q4	396.714 [395.106, 398.321]	1053.663 [1046.581, 1060.745]	231.742 [230.677, 232.806]
	M6-Q2Q3Q4-C		473.647 [470.884, 476.41]	
TMH ($\mathbf{L}_2$)	M1-Q1	**16.365** [16.171, 16.56]	**124.96** [123.541, 126.378]	**32.804** [32.443, 33.166]
	M1-Q4	**8.806** [8.608, 9.004]	**47.408** [46.894, 47.922]	**14.513** [14.343, 14.683]
	M2-Q2Q3	**25.221** [25.037, 25.405]	235.746 [233.683, 237.809]	**46.757** [46.404, 47.111]
	M4-Q2Q3	111.591 [110.188, 112.993]	634.828 [630.347, 639.309]	111.865 [110.988, 112.743]
	M6-Q2Q3Q4	214.435 [212.463, 216.408]	989.006 [982.159, 995.853]	181.077 [179.628, 182.527]
	M6-Q2Q3Q4-C		379.669 [376.796, 382.543]	

expected discounted cost together with a 95% confidence interval, after rolling out the policies for $T = 500$ steps on each episode. We also include the results obtained finding the optimal policy under information level $\mathbf{L}_3$ following policy iteration [Puterman, 2014]. We were able to solve all instances with $|\mathcal{M}| \leq 4$ using Cartesius, the Dutch national supercomputer. The averaged results are summarized in Table 6.2; a detailed investigation revealed that solutions that yield identical performance were in fact identical. We briefly discuss results for the various asset configurations:

M1-Q1 There is no information gap, and the proposed solutions match the performance and behavior of the policies found via PI. Moreover, the optimal policies coincide with the greedy heuristic (C1 and C3) and the reactive heuristic (C2).

M1-Q4 The optimal policy of the underlying MDP prescribes to either wait until the last state before failure (C1 and C3) or to delay maintenance to when the asset has failed (C2). Note that the latter policy is reactive, and indeed the proposed reactive heuristic is optimal. The best performing solutions are the TMH and *n*QR-DDQN. Since both operate under partial information of the failure probabilities, they obtain similar results by either computing an optimal TBM policy or learning via interaction.

M2-Q2Q3 When increasing the number of machines, we expect the performance gap between the $\mathbf{L}_3$ policy (obtained via PI) and the other policies to grow. In this experiment, the learned agent achieves the best performance for all cost structures. The TMH policy obtains similar performance to *n*QR-DDQN for cost structures C1 and C3. Otherwise, the TMH policy shows poor performance, since it is does not account for positive response times.

M4-Q2Q3 - M6-Q2Q3Q4 When the network size increases, the heuristics fail to cope with the increased complexity. The best performing heuristic (C1 and C3) is the TMH policy, showing that more information yields better policies, even if they are myopic. However, it consistently shows poor performance when machines with a C2 cost structure are part of the asset network. For M4-Q2Q3, *n*QR-DDQN achieves similar performance to the optimal policies under full information. It is noteworthy that the DRL agent learned to neglect one Q3 machine for M6-Q2Q3 with cost structure C1, potentially accounting for the capacity limit of the decision-maker having been reached.

M6-Q2Q3Q4-C Both the greedy heuristic and the TMH solution yield similar performance. This is due to more machines residing in the observed alert state simultaneously leading the TMH to push maintenance to meet as many deadlines as possible. Thus, the benefit of having access to a residual lifetime metric is reduced.

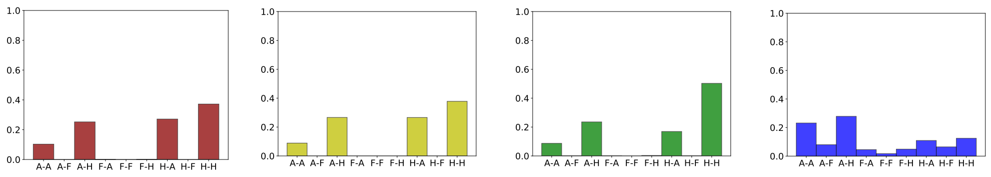
Nevertheless, the DRL agent outperforms all heuristics, implying that the learned agent is able to delay maintenance decisions more accurately.

6.7.1 Policy comparison

In this section, we focus on asset network M2-Q2Q3. This particular experiment serves as a tractable setting to understand where the proposed policies for networks containing more than one asset differ.

State visitation distribution

We select M2-Q2Q3-C1 as an instance to provide deeper insights on how frequently the various network states are visited for each policy. This latter quantity will be referred to as visitation distribution. For an instance with two machines, we can represent a censored network state using the presence of alerts (**H**: Healthy, **A**: Alert, **F**: Failed). To reduce the size of the state space, we omit the decision-maker's location and the elapsed times. Under these simplifications, we can list nine censored network states and examine the visitation distribution of each solution. Figure 6.3



(a) *n*QR-DDQN $J(\pi)$:25.139 *(b)* TMH. $J(\pi)$:25.221 *(c)* H-Greedy. $J(\pi)$:30.9 *(d)* H-Reactive. $J(\pi)$:154.074

Figure 6.3: Visitation distribution of censored network states over all episodes of problem instance M2-Q2Q3-C1.
(**H**: Healthy, **A**: Alert, **F**: Failed).

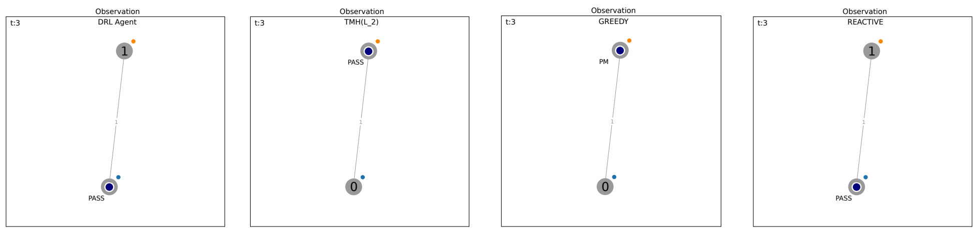
shows the visitation distribution when rolling out each policy for 512 episodes. The reactive heuristic (Figure 6.3d) visits more frequently the network states where one or both machines are in the failed state, which leads to high CM costs under the C1 cost structure. The greedy heuristic (Figure 6.3c) mostly visits network states where both machines are in a healthy state (H), implying that unnecessary costs are incurred due to excessive PM. Since the benefit of PM over CM in C1 is large, this policy yields lower costs than the reactive heuristic. Both the TMH (Figure 6.3b) and nQR-DDQN (Figure 6.3a) have similar visitation distributions since both these policies attempt to delay maintenance to the “right" time. However, the heuristic is more conservative than the learned policy, i.e., the DRL agent visits network states where the second machine is an alert state (H-A) more often. This shows that the DRL agent takes more “risk", allowing the second machine to be in the alert state slightly longer compared to the TMH.

Policy behavior

M2-Q2Q3-C2 is used as an informative instance to study the differences between policies in a given episode, in order to relate the actions taken by each policy and the performance implications.

In Figure 6.4, we present the time step $t = 3$ for a single episode of M2-Q2Q3-C2. In this example, the machine at location 0 is of type Q2, and the machine at location 1 is of type Q3. Moreover, under cost structure C2, reactive actions in the network are relatively attractive. As such, the reactive heuristic outperforms the greedy heuristic due to fewer repairs. In the example of Figure 6.4d, we observe that in the presence of an alert in machine location 1, the action taken by the reactive heuristic is to wait (PASS) until the machine fails (at $t = 6$) before traveling to that location. Under the greedy heuristic, (Figure 6.4c), the decision-maker already traveled to location 1 when the alert arrived (at $t = 2$), and the action at $t = 3$ is to perform preventive maintenance (PM) at that location, incurring the PM costs of repairing that machine. Due to the cost structure, repeatedly choosing to do PM over CM causes the greedy heuristic to accumulate higher costs.

In Figure 6.4b, we observe that the TMH makes a different decision than the greedy and reactive heuristic. Like the greedy heuristic, the TMH policy traveled to location 1 when the alert arrived. However, it decides to wait (PASS) and postpone maintenance as it favors corrective maintenance under cost structure C2. Like the reactive heuristic, the TMH performs a CM action at $t = 6$ after the machine has failed. In Figure 6.4a, we observe that given the same hidden network state, the DRL policy prescribes to remain at location 0 until $t = 4$, implying that the agent has learned that the machine at location 1 needs *at least* 3 time steps to transition to the failed network state. Thus, the DRL agent chooses a less proactive behavior than the



(a) *n*QR-DDQN. $J(\pi)$:202.311

(b) TMH. $J(\pi)$:235.746

(c) H-Greedy. $J(\pi)$:306.366

(d) H-Reactive. $J(\pi)$:283.619

Figure 6.4: Time step $t = 3$ of one episode of M2-Q2Q3C2 (best seen in color) and average performance under each policy. *Gray* nodes in the graph represent the machine locations numbered 0 (Q2) and 1 (Q3). The decision-maker is a *navy blue* dot, and PASS, MOVE, PM and CM, correspond to wait, move to another location and preventive/corrective maintenance actions. Alerts are small *orange* dots on top of machine nodes, otherwise *blue* when no alert is present.

TMH while also postponing maintenance. In fact, at $t = 4$, the DRL agent moves to location 0 and at $t = 6$, the agent performs a CM action.

At $t = 6$, both the TMH and the DRL policies have incurred the same costs and at $t = 12$, there are two alerts present in the network. Since both policies resulted in repairing the machine at location 0 at the same time, they observe identical alerts. At $t = 12$, both engineers are located at machine 0; however, the DRL agent travels to location 1, while the TMH waits until a machine fails. The DRL agent proceeds to perform PM at location 1 at $t = 13$, while at $t = 14$ the TMH observes a failure at location 1, moves to that location and performs CM at $t = 15$.

On average, after receiving an alert, the machine at location 1 (Q3) fails more frequently than the one at location 0 (Q2). At $t = 12$, the DRL agent prefers to travel to location 1 to avoid a failure while leaving the alert at location 0 unresolved. On the other hand, the TMH fails to balance the importance of the failure time distributions in this instance. Note that, at $t = 15$ and afterwards, it prefers to postpone travel and preventive maintenance actions while waiting at location 0, resulting in higher costs as it rushes to repair failed machines at both locations. For the remaining experiments, we observe that the DRL policies can manage the trade-off of costs and prioritize machines more efficiently than the other heuristics. The complete list of experiments comparing the proposed heuristics acting on the same environment is available online[3].

6.7.2 Managerial insights

For larger networks, characterizing the behavior of the learned policies via a set of heuristic rules becomes challenging as such heuristics depend on more complex rules covering more machines, alerts and elapsed times. In general, we observe that the DRL agent moves proactively to healthy assets which carry high economic risk and learns to delay preventive maintenance adequately, sometimes incurring several extra downtime penalties. This implies that proactive handling on risky assets in a complex and uncertain maintenance environment pays off. Furthermore, it is not always optimal to react on every alert in the network. It is important to understand the behaviour of each machine after an issued alert and to estimate the near-future consequences.

For large asset networks ($|\mathcal{M}| = 6$), we observe that in one instance the DRL agent leaves assets with frequent failures (in terms of the distribution of $T_m^{\mathbf{f}}$) unattended. This can be explained by the single decision-maker reaching its capacity limit. Thus, depending on the network size and degradation processes, a single decision-maker may not be sufficient to maintain the entire network without sacrificing reliability.

Managers can employ such insights when determining the size of repair crews, considering the number of assets and their degradation processes. There is a cap on the number of assets that one engineer can maintain effectively.

Each of the proposed methods in this chapter reflects on different assumptions regarding the observed information from the network and relates to different application scenarios: In case no historical information about the degradation of assets or the operational lifetimes is available, greedy or reactive policies can be easily implemented. If methods for estimating the remaining useful lifetime of assets exist, heuristic policies like the TMH can be employed to take into account the trade-off of costs of resolving the alerts in the network. When online observation is possible, or a simulation environment exists, policies can be learned via the DRL approach proposed in this work. In this case, an online system can take observations as inputs and generate instructions for a repair crew. As a result, the learned policies can inspire further heuristic developments.

[3] `http://retrospectiverotations.com/dtmpa/dtmpa.html`

6.8. Conclusion and discussion

In this chapter, we have introduced the dynamic traveling maintainer problem with alerts (DTMPA) for a network of modern industrial assets with stochastic failure times. We consider a realistic scenario where assets are part of an asset network and where alerts, triggered by real-time degradation monitoring devices, are utilized to make maintenance decisions. We have proposed a generic framework to this problem, where independent degradation processes model the alert as an early and imperfect indicator of failure. Therefore, the decision-maker only has access to partial degradation information to decide on cost-effective maintenance and travel actions.

To solve the problem, we have proposed three methods, each requiring different information levels from the alerts. When alerts come with partial distributional information about the residual lifetime, we propose a class of greedy and reactive heuristics that rank alerts based on travel times, failure times or economical value. When full residual lifetime information is available, we proposed approximating the environment with a problem instance similar to the traveling maintainer problem. This method considers the alerts to determine a schedule that minimizes the expected maintenance costs over the near future. When alerts carry only information about the time since their generation and a simulation environment is present, we can learn policies via deep reinforcement learning aimed at approximating the state-action value distributions of long-term costs.

The results show that we can effectively replicate the performance of optimal policies when no information gap exists between alerts and the real underlying degradation. When an information gap is present and the number of assets in the network is small (up to four assets), the proposed methods yield effective policies that are close to the optimal policies under full information. When the optimal policy becomes computationally intractable, the learned policies that result in the lowest total expected discounted cost balance the failure risk and the maintenance costs better than the competing methods.

Appendix

6.A. Theorems and definitions

Definition 6. *For $m \in \mathcal{M}$, let $\{\tilde{X}_m(t),\ t \geq 0\}$ denote the (observed degradation) phase process and let $\{N_m(t),\ t \geq 0\}$ denote a counting process counting the cycles of machine installments, namely $N_m(t) = \sum_{s=0}^{t} \mathbb{1}_{\{\tilde{X}_m(s)=\tilde{x}_m^{\boldsymbol{h}} \wedge \tilde{t}_m(s)=0\}}$. Here, $\tilde{t}_m(s)$ is the elapsed time since the last phase transition of machine m.*

Definition 7. *For $m \in \mathcal{M}$, let $S_m^n = \underset{t\geq 0}{inf}\{t : N_m(t) = n\}$ be the time until the start of the n-th cycle of machine m, where $n \geq 1$.*

The elapsed time since the last phase transition of machine m can now be defined as

$$\tilde{t}_m(t) = \sup_{s\in\{0,\dots,t-S_m^{N_m(t)}\}} \{s : \tilde{X}_m(t-s) = \tilde{X}_m(t)\}.$$

Theorem 3 (Discounted cost of a policy $\pi^{\mathbf{L}}$). *Let $\gamma \in [0,1]$ be the discount factor and $\tau \in \mathbb{N}$ be a maintenance deadline. The total expected discounted cost of the induced time-based maintenance policy $\pi^{\mathbf{L}}(\tau)$, denoted by $J(\pi^{\mathbf{L}}(\tau))$, satisfies*

$$J(\pi^{\mathbf{L}}(\tau)) = \frac{\bar{c}_m^{CM}\mathbb{E}[\gamma^{T_m^{\boldsymbol{a}}+T_m^f}\mathbb{1}_{\{T_m^f\leq\tau\}} \mid X_m(0) = x_m^{\boldsymbol{h}}] + \bar{c}_m^{PM}\mathbb{E}[\gamma^{T_m^{\boldsymbol{a}}+\tau}\mathbb{1}_{\{T_m^f>\tau\}} \mid X_m(0) = x_m^h]}{1 - \gamma^{t_m^{CM}}\mathbb{E}[\gamma^{T_m^{\boldsymbol{a}}+T_m^f}\mathbb{1}_{\{T_m^f\leq\tau\}} \mid X_m(0) = x_m^h] - \gamma^{t_m^{PM}}\mathbb{E}[\gamma^{T_m^{\boldsymbol{a}}+\tau}\mathbb{1}_{\{T_m^f>\tau\}} \mid X_m(0) = x_m^h]},$$

where $\bar{c}_m^{CM} = c_m^{CM} + c_m^{DT}\frac{\gamma^{t_m^{CM}}-1}{\gamma-1}$ and $\bar{c}_m^{PM} = c_m^{PM} + c_m^{DT}\frac{\gamma^{t_m^{PM}}-1}{\gamma-1}$.

Proof. The maintenance deadline induces a renewal reward process (cf. e.g, [Ross, 2014, Ch. 7]) where each new cycle starts after a maintenance period. Conditioning on the event $\{T_m^{\mathbf{f}} \leq \tau\}$ or $\{T_m^{\mathbf{f}} > \tau\}$, we find

$$\begin{aligned} J(\pi^{\mathbf{L}}(\tau)) &= \mathbb{E}[\gamma^{T_m^{\mathbf{a}}+\min\{T_m^{\mathbf{f}},\tau\}}((\bar{c}_m^{\mathrm{CM}} + \gamma^{t_m^{\mathrm{CM}}}J(\pi^{\mathbf{L}}))\mathbb{1}_{\{T_m^{\mathbf{f}}\leq\tau\}} + (\bar{c}_m^{\mathrm{PM}} + \gamma^{t_m^{\mathrm{PM}}}J(\pi^{\mathbf{L}}))\mathbb{1}_{\{T_m^{\mathbf{f}}>\tau\}}) \mid X_m(0) = x_m^{\mathrm{h}}] \\ &= \mathbb{E}[\gamma^{T_m^{\mathbf{a}}+T_m^{\mathbf{f}}}(\bar{c}_m^{\mathrm{CM}} + \gamma^{t_m^{\mathrm{CM}}}J(\pi^{\mathbf{L}}))\mathbb{1}_{\{T_m^{\mathbf{f}}\leq\tau\}} + \gamma^{T_m^{\mathbf{a}}+\tau}(\bar{c}_m^{\mathrm{PM}} + \gamma^{t_m^{\mathrm{PM}}}J(\pi^{\mathbf{L}}))\mathbb{1}_{\{T_m^{\mathbf{f}}>\tau\}} \mid X_m(0) = x_m^{\mathrm{h}}]. \end{aligned}$$

Using the linearity of expectation and sorting terms yields

$$\begin{aligned} &J(\pi^{\mathbf{L}}(\tau))\left(1-\gamma^{t_m^{\mathrm{CM}}}\mathbb{E}[\gamma^{T_m^{\mathbf{a}}+T_m^{\mathbf{f}}}\mathbb{1}_{\{T_m^{\mathbf{f}}\le\tau\}} \mid X_m(0)=x_m^{\mathrm{h}}]-\gamma^{t_m^{\mathrm{PM}}}\mathbb{E}[\gamma^{T_m^{\mathbf{a}}+\tau}\mathbb{1}_{\{T_m^{\mathbf{f}}>\tau\}} \mid X_m(0)=x_m^{\mathrm{h}}]\right) \\ &\qquad = \bar{c}_m^{\mathrm{CM}}\mathbb{E}[\gamma^{T_m^{\mathbf{a}}+T_m^{\mathbf{f}}}\mathbb{1}_{\{T_m^{\mathbf{f}}\le\tau\}} \mid X_m(0)=x_m^{\mathrm{h}}]+\bar{c}_m^{\mathrm{PM}}\mathbb{E}[\gamma^{T_m^{\mathbf{a}}+\tau}\mathbb{1}_{\{T_m^{\mathbf{f}}>\tau\}} \mid X_m(0)=x_m^{\mathrm{h}}]. \end{aligned}$$

Isolating $J(\pi^{\mathbf{L}}(\tau))$ on the l.h.s. proves the result. □

6.B. Traveling maintainer heuristic algorithms

Algorithm 1 Traveling Maintainer Heuristic

```
1: procedure CONSTRUCT INITIAL SCHEDULE
2: Input: $\sigma \in \Sigma_{\mathcal{M}_t}$
3:     Set $t^\sigma(1) = t$                                                  ▷ Current time
4:     Set $t^\sigma(2) = t^\sigma(1) + \theta_{\ell(t),\sigma(1)}$            ▷ Travel to first machine
5:     Set $j = 3$
6:     for $i = 1, \dots, |\mathcal{M}_t| - 1$ do                              ▷ Construct greedy schedule
7:         Set $t^\sigma(j) = t^\sigma(j-1) + \mathbb{1}\left\{t^\sigma(j-1) < \tilde{T}^{\mathbf{f}}_{\sigma(i),N_{\sigma(i)}(t)}\right\} t^{\mathrm{PM}}_{\sigma(i)} + \mathbb{1}\left\{t^\sigma(j-1) \geq \tilde{T}^{\mathbf{f}}_{\sigma(i),N_{\sigma(i)}(t)}\right\} t^{\mathrm{CM}}_{\sigma(i)}$   ▷ Schedule PM or CM
8:         j = j+1
9:         Set $t^\sigma(j) = t^\sigma(j-1) + \theta_{\sigma(i),\sigma(i+1)}$
10:        j = j+1
11:    end for
12:    Set $t^\sigma(j) = t^\sigma(j-1) + \mathbb{1}\left\{t^\sigma(j-1) < \tilde{T}^{\mathbf{f}}_{\sigma(|\mathcal{M}_t|),N_{\sigma(|\mathcal{M}_t|)}(t)}\right\} t^{\mathrm{PM}}_{\sigma(|\mathcal{M}_t|)} + \mathbb{1}\left\{t^\sigma(j-1) \geq \tilde{T}^{\mathbf{f}}_{\sigma(|\mathcal{M}_t|),N_{\sigma(|\mathcal{M}_t|)}(t)}\right\} t^{\mathrm{CM}}_{\sigma(|\mathcal{M}_t|)}$   ▷ End time of last maintenance action
13: Output: $\mathbf{t}^\sigma = \{t^\sigma(j) \mid j = 1, \dots, 2|\mathcal{M}_t| + 1\}$
14: end procedure
```

Algorithm 2 Traveling Maintainer Heuristic

```
1: procedure OPTIMIZE A SCHEDULE
2: Input: $(\sigma, \mathbf{t}^\sigma)$
3:     Set $\mathbf{t}^\sigma_{\mathrm{opt}} = \mathbf{t}^\sigma$, $i = |\mathcal{M}_t|$
4:     Set $\delta = \mathbf{t}^\sigma_{\mathrm{opt}}(2|\mathcal{M}_t|) - \tilde{T}^{\mathbf{f}}_{\sigma(|\mathcal{M}_t|)}$     ▷ Possible delay last repair
5:     while $\delta > 0 \wedge i > 0$ do
6:         Set $\mathbf{t}^\sigma_{\mathrm{opt}}(2i) = \mathbf{t}^\sigma_{\mathrm{opt}}(2i) + \delta$             ▷ Delay repair
7:         Set $\mathbf{t}^\sigma_{\mathrm{opt}}(2i+1) = \mathbf{t}^\sigma_{\mathrm{opt}}(2i+1) + \delta$       ▷ Delay next travel
8:         Set $i = i - 1$
9:         Set $\delta = \min\left\{\delta, \mathbf{t}^\sigma_{\mathrm{opt}}(2i)) - \tilde{T}^{\mathbf{f}}_{\sigma(i)}\right\}$    ▷ Possible delay previous repair
10:    end while
11:    Compute $c(\sigma, \mathbf{t}^\sigma_{\mathrm{opt}}) = \sum_{s=t}^{\mathbf{t}^\sigma_{\mathrm{opt}}(2|\mathcal{M}_t|+1)} \gamma^s C_{\pi^{\mathbf{L}}_s(\sigma,\mathbf{t}^\sigma_{\mathrm{opt}})}(h_s)$
12: Output: $(\sigma, \mathbf{t}^\sigma_{\mathrm{opt}}, c(\sigma, \mathbf{t}^\sigma_{\mathrm{opt}}))$
13: end procedure
```

6.C. *n*-step quantile regression double Q-Learning algorithm

Algorithm 3 *n*-step Quantile Regression Double Q-Learning (*n*QR-DDQN)

```
 1: Initialize replay memory $\mathcal{D}$, $n$, $E$, $N_B$, $T$, $N$, $\alpha$, $P$, $\lambda$, $\epsilon$.
 2: Initialize functions $\psi_w$, $\psi_{\bar{w}}$ with random weights $w = \bar{w}$
 3: for episode $= 1, \dots, E$ do
 4:     Sample initial observable state $o_0$
 5:     for $t = 0, \dots, T-1$ do
 6:         Select a random action $u_t$ with probability $\epsilon$
 7:         otherwise select $u_t = \arg\min_u \frac{1}{N} \sum_{i=1}^{N} \psi_w^i(o_t, u)$.
 8:         Execute $u_t$, observe $c_t$ and $o_{t+1}$
 9:         Store transition $(o_t, u_t, c_t, o_{t+1})$ in $\mathcal{D}$
10:         if $t \geq n$ then
11:             Sample $\left\{ \left(o_j, u_j, C^{j:j+n}, o_{j+n}\right) \right\}_{j=1}^{N_B}$ from $\mathcal{D}$
12:             $y_j^{i'} = \begin{cases} C^{j:j+n'} & \text{if } n' \leq n \text{ and } o_{j+n'} \text{ terminal} \\ C^{j:j+n} + \gamma^n \psi_{\bar{w}}^{i'}(o_{j+n}, u^*) & \text{otherwise} \end{cases}$
13:             where $u^* = \arg\min_{u'} \sum_{i=1}^{N} \psi_w^i(o_{j+n}, u')$.
14:             Take a gradient step acc. to Eq.(6.10) w.r.t. $w$
15:         end if
16:     end for
17:     Every $P$ episodes, update $\bar{w} = w$
18: end for
```

6.D. Deep reinforcement learning hyperparameters

During training of nQR-DDQN we employ a number of hyperparameters listed in Table 6.3.

Table 6.3: nQR-DDQN Hyperparameters

Hyperparameter	Description	Value
T	length of an episode	500
$\lvert\mathcal{D}\rvert$	size of the replay memory	100000
E	number of episodes	2000
P	number of episodes to update target NN	30
N_B	mini-batch size	32
γ	discount rate	0.99
e_r	exploration fraction	0.90
ϵ_i	initial exploration probability (linear decay over Ee_r episodes)	0.1
ϵ_f	final exploration probability	0.005
λ	learning rate	5×10^{-4}
n	steps to bootstrap in the n-step loss	5
α	n-step loss weight	1
κ	Huber loss weight	1
L	number of NN layers	4
N	number of quantiles in QR	51
M	number of machines in a network	$\{1, 2, 4, 6\}$
d^l	hidden dimension of layer $l = 1, \ldots, L-1$	64
$\sigma^1(x)$	linear embedding layer $l = 1$	x
$\sigma^l(x)$	activation of hidden layer $l = 2, \ldots, L-1$	$\max(0, x)$

7 Conclusion

In this thesis we studied the optimization of information acquisition for decision-intensive processes. As discussed in the introduction (Chapter 1), we considered four different research problems. In Chapter 2, we have introduced the Optimizable DIP approach. This approach is able to transform a business process model into an optimization problem, more specifically, a Markov Decision Process. This MDP can be solved using existing techniques and for a large case study we show the effectiveness of the whole approach. In Chapter 3, we make use of the natural hierarchy in companies to decompose large DIPs into smaller problems with subdecisions. The decomposed DIPs can be solved using the ODIP approach of Chapter 2 leading to policies for DIPs that would in total be intractable. In Chapter 4 we focused on the optimization part of the approach where we compare the optimal policy to heuristics that simplify the complexity of the policy. This simplification makes it easier for human decision makers to understand the proposed actions of the policy. Moreover, we introduced a case study inspired by the Quotation Optimization process of Fokker Services. In Chapter 5, we have introduced parallel information collection. The parallel information collection allows to set a constraint on the cumulative calendar time (throughput time) for the process to finish. We have used DRL algorithms to allow to solve complex DIPs that have become intractable also due to the parallelism. This helps the decision maker in improving the decision strategy and also allows to follow more naturally the use of process modelling according to BPM. In Chapter 6, we optimized the traveling and maintenance policy of an engineer in a network of assets, based on a Philips problem. We showed how DRL can be used to solve the problem and compared it to a set of heuristics. This chapter introduces a real DIP and uses a newly developed model to find the best policy for it.

In Section 7.1 we discuss the main results of this thesis. We conclude the thesis with Section 7.2 by providing future research directions.

7.1. Main results

In Chapter 2, our research goal was to define an approach that guides decision makers in making a decision in DIPs in the most efficient and effective way. We have introduced a specific way of modeling DIPs by introducing a subclass of Optimizable DIPs (ODIPs). These ODIPs are modeled using the process modeling language of CMMN. Based on this specific way of modeling we are able to derive a new approach that fills a gap in the current literature where the approach focuses specifically on the optimization of DIPs. By structuring the information that can be collected in the DIP using the notion of an Information Structure, we are capable of creating an objective that needs to be optimized. From the CMMN structure of the process we can derive the constraints on which information can be collected at what cost. Together, this gives enough stability to solve the model using a stochastic dynamic program. In an example case based on the quotation process of Fokker Services, we show the profit improvement due to the approach.

The approach in Chapter 2 is not able to handle large and complex structures in a business process. That is why in Chapter 3, we use the natural hiearchy of DIPs to decompose the optimization problem according to the ODIP approach into smaller solvable subproblems. We have introduced a new view on how CMMN and DMN can be complementary when trying to optimize complex DIPs. Especially the Decision Requirement Graph (DRG) that models the domain of decision by showing the dependencies between different subdecisions, input data and decision models. By modeling the decision hierarchy of a complex DIP with DRG, we are able to use this model to decompose a complex ODIP into a set of smaller ODIPs that can be solved individually. The hierarchy and dependency of the different decomposed ODIPs determines in which order the optimization should happen.

The decomposition of a complex ODIP and the optimization of it is captured in the approach that has been introduced. We have shown for an example case how the approach works and that it allows to optimize the policy in a fraction of the time that the ODIP approach of Chapter 2 takes.

When we look at the optimization methods that are used in the previous two chapters, it is not always possible to understand why specific decisions are optimal. Moreover in specific scenario's such as a quotation process, it might be very much appreciated by human decision makers to understand why specific decision is taken. Therefore, we aim in Chapter 4 to find the best policy when optimizing a quotation process using the Dynamic Information Acquisiton (DIA) model. The best policy might not only be optimal in terms of profit but also in terms of usability for the human decision maker. In the chapter we introduced the DIA model as a general mathematical modeling for DIPs. More specifically, for the a quotation process of a repair shop we introduced an implementation of the DIA model which we called the Quotation Optimization (QO) model. For the DIA model, we used a stochastic dynamic program (SDP) to solve it. Using a backward recursion technique, we showed how a DIA model can be optimized. Furthermore, we have shown that for specific characteristics in the DIA model it is possible to derive an optimal order of how the information should be collected. Finally, based on an instance of the QO model we show how the SDP finds the optimal policy. But we also use this SDP with some additional constraints to find a few heuristic policies that decrease the complexity of the policy for human decision makers. We are able to show, using a full factorial experiment, that fixing the order of information collection has a minimal effect on the expected profit of the process, while fixing the set of attributes completely has a negative effect on the expected profit.

An important extension in Chapter 5 of the model in Chapter 4 is to include the use of parallel tasks. This introduces a way of throughput time in the model next to the costs for using working time of an employee. The new model is a time-constrained model and requires smarter algorithms that can find policies for very large problems which optimize both throughput time and effective invested time. We refer to the new model as a throughput time constrained DIA model. We list a number of conjectures which are used as properties that simplify the optimization algorithm. We also introduce an extended QO model with time constraints. Quotation processes are an excellent example of DIPs that often have a time constraint to come up with a decision. Based on the DRL algorithm by van Jaarsveld [2020], we introduced an algorithm that uses model specific properties to improve the performance of the DRL algorithm. We show for a small example how well the DRL algorithm performs compared to the optimal solution. For a large time constrained QO model, we show the significant improvement that is made versus the base algorithm. This base algorithm does not collect information and optimizes the final decision based on the completely unkown situation. Furthermore, we have shown the effect of the optimization of a time constrained DIA model depending on the time constraint. We show the expected profit improvement when the time is increased. It is important to mention that Chapter 3 is also introducing a method that tries to solve more complex DIPs. However, this method is incapable of dealing with parallel information gathering. In that chapter, the hierarchy requires a sequential optimization of subprocesses, which is the exact opposite of what we do in Chapter 5.

The goal of Chapter 6 is to find an effective policy to the Dynamic Traveling Maintainer Problem with Alerts. This problem is a complex mix of routing, scheduling and predicting optimization. Based on a case study from Philips, we built a complete framework that models a DTMPA. For each of these three individually there is enough research to find strong policies. But in this case, the combination of the three causes the search for an good policy to be complex. We compare in this chapter three different approaches that can help in finding strong policies. Moreover, to fully understand the basic results of these approaches we have to fix some of the variables in the problem. The three approaches exist of 1) a heuristic that combines independent greedy heuristics for routing, scheduling and failure prediction, 2) a heuristic that first estimates failures to remove all uncertainty in the problem and then uses a smart algorithm that tries to solve the complex but deterministic problem, 3) a Deep reinforcement learning algorithm that is offered the complete set of parameters of the problem such that it can learn a policy which combines the dependencies in the mix of optimization problems. We show that the DRL solution is most consistent in finding strong policies for different DTMPA problems. For networks of up to 6 machines we are able to find results and make comparisons between the proposed policies and for networks of up to 4 machines we can compare the policies with the optimal result of the model as an Markov Decision Process. The results show for different gaps of information that the methods yield effective policies with a highest score for the DRL solution.

The main goal of this thesis was to bridge the gap between business process management and operations management by introducing new ideas and solutions that help decision makers turn (complex) DIPs into optimized policies which they can use to run these DIPs. What we have shown in the different chapters is that based on logically defined languages such as CMMN it is possible to model DIPs and turn them into optimization problems. Furthermore, we have shown that such optimization problems very quickly become too complex to solve exact, but using methods such as DRL or simplified heuristics we are able to show the value still of applying this optimization. Besides improvements in the optimization methods we have shown that modeling the DIPs in a smart way, using for example hierarchy, makes it possible to simplify the optimization problem. It is important to mention that, although we did introduce some additional notation for the CMMN notation, overall the framework of CMMN, bringing together process and data, is very powerful. In Chapter 3 we also show that the combination of DMN and CMMN allowd for even more powerful solutions. Hopefully, this thesis can act as the start and the inspiration for a next generation of business processes that is able to control the full holistic view as we introduced in Figure 1.1.

7.2. Future Research

In this thesis we have introduced a new view on optimizing DIPs consisting of both a BPM side and an OM side. In a world where data becomes increasingly available, this research can work on the next step in the transition from producing data to using data. To complete this thesis, we discuss some future directions of research. We separate these future directions into three categories:

- Modeling extensions in process modeling
- Modeling extensions in optimization
- Processes of application
- Fields of application

Both in the direction of operations management and information systems, we still see possibilities to improve or extend our way of modeling.

Following up on the results of Chapters 2 and 3, we think there are further ways of extending the current ODIP model. For example, using state charts it has been shown how to use hierarchy to model parallelism. If we would be able to model complex DIPs with such a technique and then use the decomposition technique of Chapter 3, we have a direct benchmark for the DRL solution of Chapter 5. Another important direction of research would be the informing of the decision maker. For this thesis, we are able to show the expected profit for each scenario. The decision maker can only choose to maximize this profit. We would like to look into a technique that can also calculate a measure of risk or deviation that makes the decision maker aware of the criticality of the decision. Continuing on the demo tool that we have built for Chapter 2, we see two important future extensions. First, the introduction of risk information in the tool and how to visualize this to the decision maker. More importantly, we think there is the possibility of building a tool that takes a correctly built DIP CMMN file as input and can automatically derive the stochastic dynamic program from that file supplemented with specific data.

Regarding the optimization approach in the different chapters we still see further possibilities of research in terms of more realistic modeling. For example, we think of information attribute dependencies, where one information attribute is already revealing part of the information of another information attribute. We think it is also possible to extend the model with optimization under imposed conditions (sustainability conditions) or the use of continuous random variables, in which case a DRL solution seems necessary. Moreover, where we focus now on maximizing the expected profit of the process, there will be many environments such as healthcare where a more robust approach that focuses on preventing worst case scenarios would be useful.

It remains complex to derive analytical insights for these DIP optimization problems since the current models. The current models are too general to be able to construct proofs. Although here is already research and proofs for DIP problems where the information structure is a linear function, we see further possibilities to extend this for other types of functions using copula where the joint distribution of multiple variables in the process can be written as univariate marginal distributions. A second approach is more inspired from the field of decision theory, where we use stochastic ordering to show the stochastic dominance of some of the attributes. By restricting the set of possible reward functions, we might be able to derive an order of collection such as we did for a very specific case in Chapter 4.

In this thesis we focused specifically on the optimization of DIPs in the field of maintenance and reliability. Moreover, the main goal of the large cases was always to optimize the bid that is offered in a tender while minimizing the time needed to find that bid value specifically for a scenario of a component repair contract in a repair work shop. However, within the field of maintenance optimization we see many other cases where our general framework can be applied. For example, in a job shop producing one of a kind items to make a price offer for which they can produce an item. Especially in an area of projects, where each job is different than any other, it is crucial to be able to estimate the value of the job correctly based on the available information. In the public domain, we see opportunities for using our framework when projects are being tendered. This tender procedure typically requires a company to estimate the costs correctly beforehand to win the tender while being profitable.

There are currently three really different application domains where we think our models could be applied as well. First, we believe that next to industry also the healthcare sector is in a data and knowledge transition. The amount of information that doctors are required to assess in order to make a diagnosis is increasing. Therefore, decision-intensive process models can be deployed as a concrete advisor either in a dynamic way or by improving the static protocols. In such scenarios we will have to focus on the explainability of decisions. A second application domain is information technology. Computers are constantly pushed into their maximum computing power to solve larger problems. When performing complex algorithmic calculations, our model can be used to assist the computer in deciding what are the right tasks to do. For example, searching through a massive information database to find some register that is needed as input might be a task that can be done later or even skipped when the output of the algorithm already can be determined based on other, less consuming, information. Finally, the financial industry is eminently a sector where information is the golden goal. Having more information than your competitors allows you to make better decisions. If you are able to buy the right stocks a millisecond earlier by having a smart model decide on what information to collect first, this can make incredible differences in profit.

In the Dynamic Traveling Maintainer Problem, we have already concluded that there is a large number of extensions to the modeling that could make the problem more interesting. Possible directions are in particular a multi-agent scenario where a larger size problem can be optimized using multiple service engineers. Furthermore, the results that are learned from the DRL agent could be the inspiration for a heuristic that is capable of copying some of the learned behaviours to construct an understandable and well-performing heuristic. Besides extending the model, we see a lot of possible applications in real life that can be modeled similarly but will require their own specific constraints to be included in this model. Examples are: The replacement of cash in ATM machines, the filling of empty snack machines, or maintenance on complex machines in a network. An interesting change to the model would also be to fix the maintainer and allow the machine to move. Think for example of navy shipyards where vessels should get to.

Bibliography

F. Afrati, S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the travelling repairman problem. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, 20(1):79–87, 1986.

A. Akcay. An alert-assisted inspection policy for a production process with imperfect condition signals. *European Journal of Operational Research*, 298(2):510–525, 2022.

R. Alford, V. Shivashankar, M. Roberts, J. Frank, and D. W. Aha. Hierarchical planning: Relating task and goal decomposition with task sharing. In *IJCAI*, pages 3022–3029, 2016.

C. Andriotis and K. Papakonstantinou. Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints. *Reliability Engineering & System Safety*, 212:107551, 2021.

A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell. Agent57: Outperforming the atari human benchmark. In *ICLR*, pages 507–517. PMLR, 2020.

R. Basten and G. van Houtum. System-oriented inventory models for spare parts. *Surveys in Operations Research and Management Science*, 19(1):34–55, 2014. ISSN 1876-7354.

K. Batoulis, A. Baumgraß, N. Herzberg, and M. Weske. Enabling dynamic decision making in business processes with dmn. In *International conference on business process management*, pages 418–431. Springer, 2016.

E. Bazhenova, F. Zerbato, B. Oliboni, and M. Weske. From bpmn process models to dmn decision models. *Information Systems*, 83:69–88, 2019.

M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

D. Bertsimas, G. Van Ryzin, et al. The dynamic traveling repairman problem. 1989.

S. Bhattacharya, S. Badyal, T. Wheeler, S. Gil, and D. Bertsekas. Reinforcement learning for pomdp: partitioned rollout and policy iteration with application to autonomous sequential repair problems. *IEEE Robotics and Automation Letters*, 5(3):3967–3974, 2020.

P. Bossaerts and C. Murawski. Computational complexity and human decision-making. *Trends in Cognitive Sciences*, 21(12):917 – 929, 2017. ISSN 1364-6613.

M. M. Botvinick. Hierarchical reinforcement learning and decision making. *Current opinion in neurobiology*, 22(6):956–962, 2012.

R. N. Boute, J. Gijsbrechts, W. van Jaarsveld, and N. Vanvuchelen. Deep reinforcement learning for inventory control: a roadmap. *European Journal of Operational Research*, 2021.

S.-P. Breton and G. Moe. Status, plans and technologies for offshore wind turbines in europe and north america. *Renewable Energy*, 34(3):646–654, 2009.

D. Bromberg. Bpm for knowledge workers: The structural foundations of decision intensive processes (dips). *BPTrends*, January 2007.

P. Bülbül, Z. P. Bayındır, and İsmail Serdar Bakal. Exact and heuristic approaches for joint maintenance and spare parts planning. *Computers & Industrial Engineering*, 129:239 – 250, 2019. ISSN 0360-8352.

F. Camci. The travelling maintainer problem: integration of condition-based maintenance with the travelling salesman problem. *Journal of the Operational Research Society*, 65(9):1423–1436, 2014.

F. Camci. Maintenance scheduling of geographically distributed assets with prognostics information. *European Journal of Operational Research*, 245(2):506–516, 2015.

R. Carpenter and A. Henderson. *Keep Them Flying: Find your winning position in the MRO game*, 2008. `https://www.ibm.com/downloads/cas/0VDARVDQ`, accessed September 11, 2020.

Cartesius. Cartesius supercomputer. URL `https://www.surf.nl/en/dutch-national-supercomputer-cartesius`.

C. D. Ciccio, A. Marrella, and A. Russo. Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches. *J. Data Semantics*, 4(1):29–57, 2015.

J. Claes, I. Vanderfeesten, F. Gailly, P. Grefen, and G. Poels. Towards a structured process modeling method: Building the prescriptive modeling theory. In *International Conference on Business Process Management*, pages 168–179. Springer, 2016.

M. Compare, L. Bellani, E. Cobelli, and E. Zio. Reinforcement learning-based flow management of gas turbine parts under stochastic failures. *The International Journal of Advanced Manufacturing Technology*, 99(9-12):2981–2992, 2018.

P. Da Costa, P. Verleijsdonk, S. Voorberg, A. Akcay, S. Kapodistria, W. van Jaarsveld, and Y. Zhang. Policies for the dynamic traveling maintainer problem with alerts. *European Journal of Operational Research*, 305(3):1141–1152, 2023.

W. Dabney, M. Rowland, M. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

K. De Asis, J. Hernandez-Garcia, G. Holland, and R. Sutton. Multi-step reinforcement learning: A unifying algorithm. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

B. de Jonge and P. A. Scarf. A review on maintenance optimization. *European Journal of Operational Research*, 285(3):805 – 824, 2020. ISSN 0377-2217.

B. de Jonge, W. Klingenberg, R. Teunter, and T. Tinga. Reducing costs by clustering maintenance activities for multiple critical units. *Reliability Engineering & System Safety*, 145:93 – 103, 2016. ISSN 0951-8320.

M. De Leoni, P. Felli, and M. Montali. Integrating bpmn and dmn: modeling and analysis. *Journal on Data Semantics*, 10(1):165–188, 2021.

S. Debois, T. T. Hildebrandt, and T. Slaats. Hierarchical declarative modelling with refinement and sub-processes. In *Proc. BPM 2014*, pages 18–33, 2014.

L. Deprez, K. Antonio, and R. Boute. Pricing service maintenance contracts using predictive analytics. *European Journal of Operational Research*, 2020. ISSN 0377-2217.

C. Derman. On optimal replacement rules when changes of state are markovian. *Mathematical optimization techniques*, 396:201–210, 1963.

B. L. Dos Santos and V. S. Mookerjee. Minimizing information acquisition costs. *Decision Support Systems*, 9(2):161–181, 1993.

C. Drent, M. O. Keizer, and G.-J. van Houtum. Dynamic dispatching and repositioning policies for fast-response service networks. *European Journal of Operational Research*, 285(2):583–598, 2020.

M. A. Driessen, J. W. Rustenburg, G. J. van Houtum, and V. C. Wiers. Connecting inventory and repair shop control for repairable items. In *Logistics and Supply Chain Innovation*, pages 199–221. Springer, 2016.

M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, et al. *Fundamentals of business process management*, volume 1. Springer, 2013.

R. Eshuis. Modeling decision-intensive processes with declarative business artifacts. In *Proc. ASSRI 2018*, pages 3–12, 2018.

R. Eshuis, M. Firat, and U. Kaymak. Modeling uncertainty in declarative artifact-centric process models using fuzzy logic. *Information Sciences*, 579:845–862, 2021.

H. D. Garcia. Evaluation of data-centric process modeling approaches. *Eindhoven University of Technology*, 2011.

A. González-Ferrer, J. Fernández-Olivares, and L. Castillo. From business process models to hierarchical task network planning domains. *The Knowledge Engineering Review*, 28(2): 175–193, 2013.

A. Grudzińska-Kuna. Supporting knowledge workers: case management model and notation (cmmn). *Information Systems in Management*, 2(1):3–11, 2013.

H. K. Hall, J. C. Moore, and A. B. Whinston. A theoretical basis for expert systems. *IFAC Proceedings Volumes*, 19(17):11–19, 1986.

F. Hasić, J. De Smedt, and J. Vanthienen. Augmenting processes with decision intelligence: Principles for integrated modelling. *Decision Support Systems*, 107:1–12, 2018.

M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. L. Dean, and C. Boutilier. Hierarchical solution of markov decision processes using macro-actions. *arXiv preprint arXiv:1301.7381*, 2013.

M. J. Havinga and B. de Jonge. Condition-based maintenance in the cyclic patrolling repairman problem. *International Journal of Production Economics*, 222:107497, 2020. ISSN 0925-5273.

J. F. Hernandez-Garcia and R. S. Sutton. Understanding multi-step deep reinforcement learning: a systematic study of the dqn target. *arXiv preprint arXiv:1901.07510*, 2019.

M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

T. T. Hildebrandt, R. R. Mukkamala, and T. Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *Proc. EDOC 2011*, pages 161–170, 2011.

F. E. Horita, J. P. de Albuquerque, V. Marchezini, and E. M. Mendiondo. Bridging the gap between decision-making and emerging big data sources: An application of a model-based framework to disaster management in Brazil. *Decision Support Systems*, 97:12 – 22, 2017. ISSN 0167-9236.

Q. Hu, J. E. Boylan, H. Chen, and A. Labib. Or in spare parts management: A review. *European Journal of Operational Research*, 266(2):395 – 414, 2018. ISSN 0377-2217.

P. J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101, 1964.

T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201, 1994.

L. Janssens, E. Bazhenova, J. De Smedt, J. Vanthienen, and M. Denecker. Consistent integration of decision (dmn) and process (bpmn) models. In *CAiSE forum*, volume 1612, pages 121–128, 2016.

J. Jeston. *Business process management: practical guidelines to successful implementations*. Routledge, 2014.

K. B. Kallestrup, L. H. Lynge, R. Akkerman, and T. A. Oddsdottir. Decision support in hierarchical planning systems: The case of procurement planning in oil refining industries. *Decision Support Systems*, 68:49–63, 2014.

T. Kenbeek, S. Kapodistria, and A. Di Bucchianico. Data-driven online monitoring of wind turbines. In *Proceedings of the 12th EAI International Conference on Performance Evaluation Methodologies and Tools*, page 143–150. Association for Computing Machinery, 2019.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

D. Klabjan, W. Olszewski, and A. Wolinsky. Attributes. *Games and Economic Behavior*, 88: 190–206, 2014.

A. Kuhnle, J. Jakubik, and G. Lanza. Reinforcement learning for opportunistic maintenance optimization. *Production Engineering*, 13(1):33–41, 2019.

M. G. Lagoudakis and R. Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 424–431, 2003.

C. Lim, J. N. Bearden, and J. C. Smith. Sequential search with multiattribute options. *Decision Analysis*, 3(1):3–15, 2006.

Y. Liu, Y. Chen, and T. Jiang. Dynamic selective maintenance optimization for multi-state systems over a finite horizon: A deep reinforcement learning approach. *European Journal of*

Operational Research, 283(1):166–181, 2020.

M. A. Marin, M. Hauder, and F. Matthes. Case management: An evaluation of existing approaches for knowledge-intensive processes. In *Proc. BPM 2015 Workshops*, pages 5–16, 2015.

A. Marrella, M. Mecella, and S. Sardiña. Intelligent process adaptation in the smartpm system. *ACM Trans. Intell. Syst. Technol.*, 8(2):25:1–25:43, 2017.

J. Merel, M. Botvinick, and G. Wayne. Hierarchical motor control in mammals and machines. *Nature communications*, 10(1):1–12, 2019.

S. Mertens, F. Gailly, and G. Poels. Towards a decision-aware declarative process modeling language for knowledge-intensive processes. *Expert Systems with Applications*, 87:316 – 334, 2017. ISSN 0957-4174.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

V. S. Mookerjee and B. L. Dos Santos. Inductive expert system design: maximizing system value. *Information Systems Research*, 4(2):111–140, 1993.

V. S. Mookerjee and M. V. Mannino. Sequential decision models for expert system optimization. *IEEE Trans. Knowl. Data Eng.*, 9(5):675–687, 1997a.

V. S. Mookerjee and M. V. Mannino. Redesigning case retrieval to reduce information acquisition costs. *Information Systems Research*, 8(1):51–68, 1997b.

V. S. Mookerjee and M. V. Mannino. Sequential decision models for expert system optimization. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):675–687, 1997c.

J. C. Moore and A. B. Whinston. A model of decision-making with sequential information-acquisition (part 1). *Decision Support Systems*, 2(4):285–307, 1986.

J. C. Moore and A. B. Whinston. A model of decision-making with sequential information-acquisition (part 2). *Decision Support Systems*, 3(1):47–72, 1987.

J. A. Muckstadt. A model for a multi-item, multi-echelon, multi-indenture inventory system. *Management Science*, 20(4-part-i):472–481, 1973.

H. Nguyen, M. Dumas, A. H. ter Hofstede, M. La Rosa, and F. M. Maggi. Stage-based discovery of business process models from event logs. *Information Systems*, 84:214–237, 2019.

Object Management Group. Business process model and notation. version 2.0, 2010.

Object Management Group. Case management model and notation. version 1.1, 2015.

Object Management Group. Decision model and notation. version 1.3, 2020.

M. C. Olde Keizer, S. D. P. Flapper, and R. H. Teunter. Condition-based maintenance policies for systems with multiple dependent components: A review. *European Journal of Operational Research*, 261(2):405 – 420, 2017. ISSN 0377-2217.

M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. DECLARE: full support for loosely-structured processes. In *Proc. EDOC 2007*, pages 287–300, 2007.

J.-C. Pomerol. Artificial intelligence and human decision making. *European Journal of Operational Research*, 99(1):3 – 25, 1997. ISSN 0377-2217.

W. B. Powell. A unified framework for stochastic optimization. *European Journal of Operational Research*, 275(3):795–821, 2019.

M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

S. M. Ross. *Introduction to probability models*. Academic press, 2014.

T. L. Saaty. How to make a decision: the analytic hierarchy process. *European journal of operational research*, 48(1):9–26, 1990.

B. Sahakian and J. N. LaBuzetta. *Bad Moves: How decision making goes wrong, and the ethics of smart drugs*. OUP Oxford, 2013.

M. Sarafyazd and M. Jazayeri. Hierarchical reasoning by neural circuits in the frontal cortex. *Science*, 364(6441):eaav8911, 2019.

H. Schonenberg, B. Weber, B. F. van Dongen, and W. M. P. van der Aalst. Supporting flexible processes through recommendations based on history. In *Proc. BPM 2008*, pages 51–66, 2008.

S. Sen and A. Kumar. *Design and analysis of algorithms: A contemporary perspective*. Cambridge University Press, 2019.

I. Sid, M. Reichert, and A. R. Ghomari. Enabling flexible task compositions, orders and granularities for knowledge-intensive business processes. *Enterprise Information Systems*, 13(3):376–423, 2019.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

H. A. Simon. The sciences of the artificial, 1969. *Massachusetts Institute of Technology*, 1981.

A. Sleptchenko, M. C. van der Heijden, and A. van Harten. Trade-off between inventory and repair capacity in spare part networks. *Journal of the Operational Research Society*, 54(3):263–272, 2003.

A. Sleptchenko, H. H. Turan, S. Pokharel, and T. Y. ElMekkawy. Cross-training policies for repair shops with spare part inventories. *International Journal of Production Economics*, 209: 334–345, 2019.

J. C. Smith, C. Lim, and J. N. Bearden. On the multi-attribute stopping problem with general value functions. *Operations Research Letters*, 35(3):324–330, 2007.

A. Suchenia, K. Kluza, P. Wiśniewski, K. Jobczyk, and A. Ligęza. Towards knowledge interoperability between the uml, dmn, bpmn and cmmn models. In *MATEC Web of Conferences*, volume 252, page 02011. EDP Sciences, 2019.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

E. Topan, A. Eruguz, W. Ma, M. van der Heijden, and R. Dekker. A review of operational spare parts service logistics in service control towers. *European Journal of Operational Research*, 282 (2):401 – 414, 2020. ISSN 0377-2217.

T. Tulabandhula, C. Rudin, and P. Jaillet. The machine learning and traveling repairman problem. In R. I. Brafman, F. S. Roberts, and A. Tsoukiàs, editors, *Algorithmic Decision Theory*, pages 262–276, 2011.

O. Turetken, A. Dikici, I. Vanderfeesten, T. Rompen, and O. Demirors. The influence of using collapsed sub-processes and groups on the understandability of business process models. *Business & Information Systems Engineering*, 62(2):121–141, 2020.

R. Vaculín, R. Hull, T. Heath, C. Cochran, A. Nigam, and P. Sukaviriya. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *Proc. EDOC 2011*, pages 151–160, 2011.

H. van der Aa, H. Leopold, K. Batoulis, M. Weske, and H. A. Reijers. Integrated process and decision modeling for data-driven processes. In *Proc. BPM 2015 Workshops*, pages 405–417, 2015.

S. Van der Auweraer and R. Boute. Forecasting spare part demand using service maintenance information. *International Journal of Production Economics*, 213:138 – 149, 2019. ISSN 0925-5273.

H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

W. van Jaarsveld. Deep controlled learning of dynamic policies with an application to lost-sales inventory control. *arXiv preprint arXiv:2011.15122*, 2020.

W. van Jaarsveld, T. Dollevoet, and R. Dekker. Improving spare parts inventory control at a repair shop. *Omega*, 57:217–229, 2015.

H. E. van Staden and R. N. Boute. The effect of multi-sensor data on condition-based maintenance policies. *European Journal of Operational Research*, 290(2):585–600, 2021. ISSN 0377-2217.

I. T. P. Vanderfeesten, H. A. Reijers, and W. M. P. van der Aalst. Product-based workflow support. *Information Systems*, 36(2):517–535, 2011.

S. K. Venero, J. C. dos Reis, L. Montecchi, and C. M. F. Rubira. Towards a metamodel for supporting decisions in knowledge-intensive processes. In *Proc. SAC 2019*, pages 75–84, 2019.

S. K. Venero, B. R. Schmerl, L. Montecchi, J. C. dos Reis, and C. M. F. Rubira. Automated planning for supporting knowledge-intensive processes. In *Proc. BPMDS 2020*, pages 101–116. Springer, 2020.

S. Voorberg, R. Eshuis, W. van Jaarsveld, and G.-J. van Houtum. Decision support for declarative artifact-centric process models. In *Business Process Management Forum*, pages 36–52, 2019.

S. Voorberg, R. Eshuis, W. van Jaarsveld, and G. van Houtum. Decisions for information or information for decisions? optimizing information gathering in decision-intensive processes. *Decision Support Systems*, 151:113632, 2021.

S. Voorberg, W. van Jaarsveld, R. Eshuis, and G. van Houtum. Information acquisition for service contract quotations made by repair shops. *European Journal of Operational Research*, 305(3):1166–1177, 2023.

W. Wang. An overview of the recent advances in delay-time-based maintenance modelling. *Reliability Engineering & System Safety*, 106:165–178, 2012. ISSN 0951-8320.

M. Wiemuth, D. Junger, M. Leitritz, J. Neumann, T. Neumuth, and O. Burgert. Application fields for the new object management group (omg) standards case management model and notation (cmmn) and decision management notation (dmn) in the perioperative field. *International journal of computer assisted radiology and surgery*, 12(8):1439–1449, 2017.

A. Yousfi, A. K. Dey, R. Saidi, and J. Hong. Introducing decision-aware business processes. *Computers in Industry*, 70:13–22, 2015.

A. Yousfi, K. Batoulis, and M. Weske. Achieving business process improvement via ubiquitous decision-aware business processes. *ACM Trans. Internet Techn.*, 19(1):14:1–14:19, 2019.

Summary

Optimization of Information Acquisition for Decision-Intensive Processes

When we look at the current economy and its goals, we can say that the modern labour force is more and more shifting in the direction of knowledge-intensive and decision-intensive jobs. Such jobs are based on the capabilities of workers to make decisions based on collected knowledge or information. Decision-intensive business processes consist of a phase where information is collected before a decision is taken in the subsequent phase. In our research we focus on dynamically optimizing how much information to collect in which order (Phase I), followed by optimizing the decision (Phase II). These two phases are dependent on each other. Collecting more information in Phase I is costly and requires *efficiency*. The challenge is to choose which information to collect and, based on that new information, how to continue. Also, extra information allows to make a better decision in Phase II to improve the *effectiveness*. That is why we build models that can model the complex behaviour of the decision-intensive process and optimize such processes.

Decision-intensive jobs are an interplay between processes, information and decisions. These three can be modeled using modeling languages such as Business

Process Modeling Notation (BPMN), focusing on repetitive processes, Case Management Modeling Notation (CMMN), focusing on unique information cases, or Decision Modeling Notation (DMN), focusing on the decision part of a process. What is missing, however, is a connecting approach that takes these three views together and tries to optimize them jointly. By defining relations between process tasks and information attributes and between information attributes as parameters for a decision, we can take those three views in one optimization model. In Chapter 2, *Decisions for Information or Information for Decisions? Optimizing information gathering in Decision-Intensive Processes*, we introduce an approach with a CMMN case model as the model that connects tasks and information, and an information structure that connects information as parameters for a decision. Using this case model and information structure, we optimize the decision-intensive process for both phases: We support a decision maker in the continuous trade-off between the efficient acquisition of more information (Phase I) and cost-effective decision making (Phase II). The resulting policy can be used as a run-time recommendation tool to the decision maker.

Chapter 3, *Hierarchical decision making in Decision-Intensive Processes*, makes use of a decision hierarchy to divide large-scale decision-intensive processes into small scale 'regional' problems. This decision hierarchy can be perfectly modeled using Decision Requirement Graphs (DRG). The smaller problems are solved by setting subgoals that lead to subdecisions. These subproblems can now be seen as an independent problem that can be solved according to the methods described in Chapter 2. Such subdecisions work as input parameters for the aggregate problem in the hierarchy. We show the effectiveness of this method and how well it performs in both small size problems, where we can compare to the optimal solution, and large size problems, where we show the performance for a subpart of the full problem.

When we approach decision-intensive processes from an Operations Management perspective, we model them such that we can apply known techniques to optimize and find useful but cost-effective heuristics. In Chapter 4, *Information Acquisition for Service Contract Quotations Made by Repair Shops*, we show how dynamic programming can be used to optimize the approach that we introduce in Chapter 2. Moreover, we introduce two alternative heuristics that increase the comprehensibility for the human decision maker, while trying to stay as optimal as possible. These heuristics limit the amount of possible decisions, either by fixing the order of possible information gathering or by even fixing beforehand what information to collect. We show that a fixed order of collecting information allows for very close to optimal results. Finally, for a specific setting of this problem with normally distributed information variables, we prove there is an optimal order of collecting information.

Chapter 5 is on *Using Reinforcement Learning for Optimal Information Acquisition in Decision Processes*. We use a model-based Deep Reinforcement Learning (DRL) technique to optimize large scale decision-intensive processes. We extend the basic model from Chapter 4 with lead time constraints and parallel information acquisition, such that we can put a maximum throughput time constraint on the process. These extensions cause the state space of the problem to explode and we aim at showing the value of DRL for problems like this. For small size problems we make a comparison to the optimal solution and for larger size problems we introduce some heuristics against which we compare the performance of the DRL solution.

In Chapter 6, we focus on a specific decision problem and compare *Policies for the Dynamic Traveling Maintainer Problem with Alerts* (DTMPA). In this problem, a maintenance engineer travels around in a network of assets that require maintenance. We assume those assets are degrading according to a stochastic process but send out an alert based on sensor data when approaching a breakdown. Using the available information that this alert contains, which we define as the information level and can differ in informativeness, we look at different methods to optimize the dispatching of the engineer. For small size networks, we show how well a greedy heuristic, a new developed heuristics and an DRL agent perform compared to a solution when all information is known. We give insights on the resulting policies and show how capable the DRL agent is in learning the degradation process distribution.

Acknowledgments

The exciting journey of a PhD never was really in my sight. I felt like any of the other 200 students at my bachelor in Industrial Engineering and was sometimes even overwhelmed by the subjects in my MSc program. However, it was my thesis supervisor Maria Vlasiou who planted the little seed in my head of doing a PhD after my MSc., for which I thank her deeply. She already talked me almost into it during my final months of my master thesis, which was the position eventually taken by Mirjam. However, after some months of applying at companies and not feeling satisfied by the challenges offered, I came across a PhD position in the middle of two interesting research fields that I had encountered during my studies at TU/e. Willem was one of the supervisors, whom Maria already told me was a good fit for me and hence, that is where the adventure started. Now, five years later, I look back at the best period of my life in which I fully submerged myself in the world of academia. As ignorant as I was when I entered it, it gives me great pleasure to see what I have been able to achieve in all of these years. A vibrant research environment, such as OPAC offered me, allowed me to develop myself in every way possible. That is why I would like to express my thanks to those who have contributed to this exciting journey.

To start off, I would like to thank my two daily supervisors, Willem and Rik. The connection that Willem and I had and have, has always amazed me. His enthusiasm and determination really were of great value at many moments of my PhD. There were plenty of moments where he would just walk by to have someone to talk to, about all kind of topics. These daily little exchanges are finally what allowed me finish this PhD. Rik could be seen as the compass of my thesis. Where Willem could set off into unknown waters driven by his enthusiasm, it would always be Rik who would look at the horizon and keep focused on the goals. His calm and thoughtful way of looking at problems allowed me to achieve this PhD. I would also like to thank my promotor, Geert-Jan. Geert-Jan was not always involved in the daily steps of progress, but he knew when to show up on stage. His experience and guidance throughout the whole process allow me to look back at my PhD with a lot of happiness and pleasure.

I would like to thank also Peter, Paulo, Stella, Alp, and Yingqian. Together, we created a paper that is now Chapter 6 in this thesis. Especially Peter and Paulo, working together has allowed me to see what is the value of cooperation in academia and how easy that can lead to interesting interdisciplinary work.

I would like to thank Robert Boute, Rommert Dekker, Remco Dijkman, and Maria Iacob for taking part in my committee. It is a pleasure to have people in your committee whom you have seen or spoken to already. Robert, thanks for receiving us during the ISIR conference in 2019. It has been one of the most exciting conferences in my PhD. Remco, thanks for your friendly exchanges in the Atlas building. Rommert, it has always been a pleasure to listen to some of your talks at conferences. Maria, thanks for the efforts you made in a difficult period.

I would like to thank also the consortium in which this PhD project took place. Working with interesting companies such as Fokker Services, NS or Philips really showed me the value of bringing research into practice. Wouter van Dis, Wouter van Heeswijk, Bob Huisman, Verus Pronk, and Jan Korst, thanks for all your valuable input during our many consortium meetings. It is rather inspiring to see how you were able to provide us with long lists of problems that would require research.

Thanks also to Christel, Jolanda and José. Your availability and help in the daily running of a research department is of great value for all the staff.

A special thanks to Claudine. The nice evenings at your place together with Miguel will always stay in my mind. The endless talks we had were maybe not always production-enhancing, but the more valuable. It is nice to have colleagues with

whom you can discuss real life other than work.

A big thanks also to Mirjam. She was the steady factor in all of these PhD years. Whichever day it was, she was there at 7:30 and would be there until 16:30. Apart from the proofs that we derived together for classes, your calmness might have been more of use to me than you were aware of yourself. It gives me great pleasure to stay in contact with you and still work with you.

Even though the last two years, mainly due to COVID, we were less in contact, I would like to thank Joni also. My roommate during the first year that we were still in Paviljoen and my partner in crime, as I felt it. I can really say that thanks to you the beginning of my PhD was a joy.

I would like to give a warm thanks to Zümbül. It was such a pleasure to have you around in the office. You really were someone where the door is always open for a nice talk or a coffee. Your presence in both happiness and sadness makes me feel you as a dear friend.

There are four people that I cannot forget to thank. Sami, Virginie, Afonso, Claudia, many thanks for all the wonderful moments we had in these years. Even though we all went our own ways and even though many people cannot understand how you can have friends so far away, I still cherish the friendship we have.

Thanks also to Sjors and Yeşim. Sjors, your thesis acknowledgements will forever stay in my mind. Our ski trip was one of the best weeks of my life. I really hope you will find your happiness with Yeşim in Eindhoven.

A big thanks also to Şifa and Patrick. Due to COVID, I feel my PhD was cut into multiple parts, and you were part of the last OPAC stage. Still, I enjoyed our time very much and hope to keep you close as friends for the future.

I would also like to thank Joost, Wendy, Volkan, Ragnar, Laura, Bram, Taher, Loe, Karel, Kaj, and Shaunak. Each of you played a different role during my years as a PhD. But, while writing this part, I can clearly think of many happy moments I enjoyed with each of you.

I would like to thank my whole family for all the moments of interest and encouragement. Specifically, my parents, Jaap and Elsemieke, thanks for the never ending pushes in making the most out of what is given to us. Even though I am now living far away, I will forever hold you close. Friso and Geralde, thank you for your patience with a brother who always thinks he knows everything better and might never be able to drop that.

Finally, an enormous thanks to Paciane for all your love and support. It was a great gift to find you in Taizé during this PhD. All the days that we spent together in Paris during the COVID lockdowns really made me learn what a wonderful person I have found in my life. I won’t let you go anymore.

About the author

Simon Voorberg was born in Sliedrecht, the Netherlands, on June 9th 1992. He finished his pre-university education at the Gereformeerde Scholengemeenschap Randstad in Rotterdam in 2010. In 2014, he obtained his BSc in Industrial Engineering from Eindhoven University of Technology with a minor in Computer Science. After that, he pursued a MSc degree at the same university in the subject Industrial and Applied Mathematics with a specialization in Statistics, Probability and Operations Research. During this master he cooperated with the companies FloraHolland and Tech Data.

In November 2017, he started his PhD research within the Operations, Planning, Accounting and Control group at the department of Industrial Engineering and Innovation Sciences at Eindhoven University of Technology. Under supervision of dr. Willem van Jaarsveld, dr. Rik Eshuis and prof. dr. Geert-Jan van Houtum, he worked on bringing together the fields of Information Systems and Operations Management by looking into the optimization of knowledge-intensive business processes. During this research, he worked with companies such as Fokker Services, Philips and Dutch Railways in a research project supported by the Dutch Science Foundation (NWO).

Since September 2022, Simon has a position as an assistant professor in the department of Information Systems, Supply Chain Management and Decision Support department at NEOMA business school in Rouen.