Leiden University

ICT in Business and the Public Sector

Resource Allocation Optimization Through Process Mining Within a Business Environment

Name:	Kolenbrander, Marcel
Student-no:	1653415
Date:	31st of October, 2022
1st supervisor:	Dr. Y. Fan
2nd supervisor:	Prof. Dr. T. H. W. Bäck
Data and Code repository:	https://github.com/MarcelKolen/process-mining-resource-allocation-optimizer

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University

Niels Bohrweg 1 – 2333 CA Leiden – The Netherlands

Abstract

This thesis introduces a novel multi-objective resource allocation optimization framework using a divide and conquer approach operating on a zero-knowledge basis (meaning that the optimizer gets no additional input other than a process log). The multi-objective optimization method uses process mining as part of its mathematical optimization model construction element to gather all its insights to perform resource allocation optimizations from a log. The primary goal of this optimizer is to offer end-users optimized resource allocation suggestions on a multi-objective Pareto optimum front, based on historical process event data logs, and to offer future researchers a framework to expand on to test new optimization objectives, process constraints and external behaviour model variables. The optimizer should work within the confines of an event data log and should not require any prior knowledge about a process or the (cost) behaviour of its resources, nor any other external inputs other than the event data log, henceforth the zero-knowledge basis. Optimized resource allocation suggestions can help advance the performance of a process and is therefore an important and relevant field of in the practise of process management. Not only can this resource allocation optimization method be used for business processes, but it can also be used for supply chain processes. Whilst there are other resource allocation techniques, these techniques either only optimize parts of a process, require prior knowledge about the behaviour of resources, are limited to using linear models only, or require the construction of a stochastic optimization model. This method differentiates itself with its whole process approach, its expandable dynamic resource cost modelling capabilities (including non-linear scenarios), and its deterministic characteristics.

KEYWORDS: Resource allocation, Optimization, Minimization, Pareto optimum front, Process mining, Inductive mining, Business processes, Supply chain processes, Process simulation, Zero-knowledge, Divide and conquer

Table of Contents

1 Introduction

In modern business environments and technology and data driven supply chains, process mining can play an instrumental role in understanding the integral parts of actual work being done. Process mining is driven by business and/or supply chain event data, essentially logs of activities [1]–[3]. When process mining is used, every triggered or activated event within a business process or a supply chain is recorded in a logging system.

Traditionally, the event data is used to generate insight into existing processes by representing these processes into process-maps and runtime insights [1], [2]. However, in a rapidly digitalizing world, where business environments are increasingly more driven by decisions-by-data [4], one could wonder whether process mining can be used with more goals, such as business process management, in mind.

1.1 Problem Statement & Project Definition

The next step above simple process discovery and visualization, is business process management in the form of process optimization through the analyses of process event data. In the field of process mining, there already exist several techniques for identifying bottlenecks [5]–[7] in a process. By knowing where a bottleneck occurs, efforts and resources can be allocated to alleviate this bottleneck. A reasonable assumption could also be made that through process mining, and the analysis of event data, alleviation suggestions could be generated for said bottlenecks.

However, this initial approach of alleviating a bottleneck is flawed because of the definition of a bottleneck. According to the definition of a bottleneck there can only be one bottleneck in a process [8]. By extension, alleviating one bottleneck simply creates another bottleneck. Moreover, solving one bottleneck might create a worse bottleneck elsewhere in a process.

Instead of searching for and alleviating bottlenecks in a process, it is perhaps a better approach to optimize for an entire process. This requires converting a process into a mathematical model which can be subject to optimization by means of one (1) or more input parameters.

As a process is transformed into a mathematical model which can be optimized by tuning input parameters, lastly the input parameters need to be determined. A process could have a variety of different parameters interacting with one and another which all impact the outcome of a process. However, two (2) of the main elements in a process which interact with one (1) and another are activities and resources [1], [3]. With these two (2) elements being the most prominent in a process, this thesis will focus on developing an optimization method whereby the allocation of resources to activities is optimized.

The next few subsections will be devoted to further formulating the definitions required to develop a process optimization framework.

1.1.1 Optimization definition

The term optimization has been used several times now, however, a clear and restricted definition of optimization has not yet been determined. There are several ways, or key performance indicators (KPIs), by which to optimize a process. Throughout this thesis, optimization is determined to be optimization by means to better the following three (3) KPIs, or, later known as, objective functions:

- **THROUGHPUT TIME** how long, in terms of time, does a step in a process, or the entire process take, to complete;
- **WAIT TIME** how much time exists between the *end* and *start* of steps in a process;
- **COST** how much does it cost, in terms of monetary amounts, to execute a step in a process, or the entire process.

Optimizing these specific KPIs translates to the reduction of these KPIs, for example reducing cost, as compared to a baseline. This baseline could for example be a found average across a dataset. For other

KPIs optimization might mean increasing the KPI value above a certain baseline. Depending on the KPI, the optimization problem essentially becomes a minimization or a maximization exercise.

These three (3) KPIs partially serve an illustrative role in this thesis. By no means should these three (3) KPIs be seen as exhaustive. Any other KPI, or objective, which can be mathematically described could be applied.

1.1.2 Process definition

Besides the definition of optimization, another definition which has been and will be used more frequently is *a process*. The question then arises, what is the definition of a process?

In this thesis, a process is defined as: *"A set of activities, which interact with one and another, and which may be performed in a various, but consistent, set of ways to achieve a certain goal"*. This definition is derived from the ISO 9001 definition of a process: *"A process: set of interrelated or interacting activities that use inputs to deliver an intended result"* [9].

However, as this thesis focusses on the allocation optimization of resources to activities, this definition is will be further extended by introducing resources into the process definition. This creates the following definition for a process: *"A set of activities, which interact with one and another, and which may be performed by a set of resources in a various, but consistent, set of combinations to achieve a certain goal"*.

An example of a process, in the form of a Business Process Model and Notation (BPMN) diagram, is provided in ***Figure 1***.

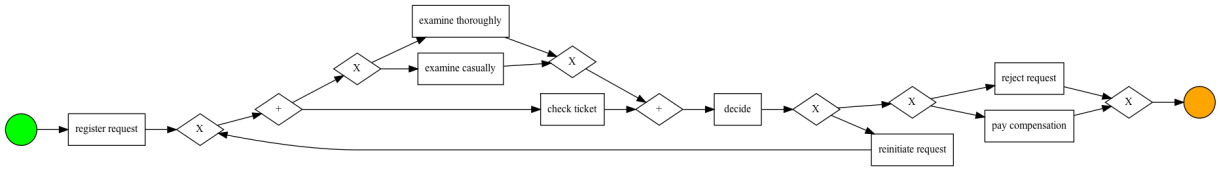


Figure 1 – Process model example 0, represented in a BPMN schema. Process based on a dataset example curtesy of PM4PY [10], [11]. *Please refer to **10.1** for a larger version.*

1.1.3 Whole process optimization

As said before, instead of optimizing by focusing on solving one (1) bottleneck at a time, or optimizing sub-processes, it might make more sense to look at how to optimize a process as a whole. In the previous two (2) subsections definitions were given for what is to be optimized for, the KPIs, or rather objective functions, and what we can optimize on, a process, more specifically the allocation of resources in that process. There are now only two (2) last parts remaining, which are, how are these objective functions formulated and should a process be converted into a mathematical model?

The answers of these two (2) questions go hand in hand with on and another when trying to answer them to finalize the optimization strategy which will be used to optimize resource allocations in this thesis. The result of an objective function over a process should essentially be a (complex) sum of activity behaviours. These activity behaviours are dependent on their respective resource allocations. These behaviours of resources on activities have to be formulated using modelling strategies, such as for example regression modelling, based on the available process data log.

As the objective functions are to be approached as sums of activity behaviours, a definition of how this sum is to be formulated should be given. This is precisely related to how a process is to be mathematically modelled. There are essentially two (2) main approaches. The first approach, which will also be covered in **Current resource-activity allocation optimization methods** is to convert the process into a single large stochastic mathematical model. There the components of the process are approached using probabilistic elements. This stochastic model is optimized once, and the result of the

single optimization is the final result. A different, novel, method, which will be the method used in this thesis, is a divide and conquer approach where a process is divided up in multiple simpler, pruned, and deterministic models. These multiple models are then individually optimized, and their individual results are merged into one (1) generalistic result.

1.1.4 Problem Statement

The previous subsections were used to introduce several components and definitions regarding process optimization. An outline of the strategy for the process optimization which will be used in this thesis has been outlined as well. Here the different components are brought together and summarized as one (1) large problem/strategy statement.

In this thesis process optimization will be done by means of resource allocation optimization. It will use a divide and conquer approach where processes are divided up in multiple simpler, pruned, and deterministic models. The optimizer works on a minimization or a maximization basis, based on the chosen KPI/objective function. Objective functions are depended on the behaviour or activity-resource combinations, which in turn will be modelled using the input event-data using regression strategies. The expected outcome is a set of optimal combinations of resource and activities.

1.2 Research Objectives

With the problem statement defined, the research objectives and questions can be defined. As stated before, this thesis aims to develop a divide and conquer style process optimizer which optimizes by trying to find a most optimal resource-activity allocation, using process event data analysis. However, in order to be able to perform proper process analysis, activity-resource behaviour, and finally process optimization, suitable data is required.

With this, the first research question can be defined as follows:

> *"What are the (minimum) data requirements, in terms of format, contents, and quantity, in order to allow an optimizer to perform resource-activity allocation optimizations?"*

With the question of data set out, the next question can be developed. This next question and objective revolve around putting the available data to use and find an optimal resource-activity combination. Through process mining, an event-based dataset should be transformed into a process map. This process map is then to be divided into simpler deterministic process variants, and in these pruned process maps, the optimizer should try to find combinations of resources and activities which can give the best average result in terms of throughput time, wait time, and cost. With this in mind, the second research question can be formulated:

> *"How can process mining be used to optimize the allocation of resources to activities, using a divided and conquer approach?"*

1.3 Thesis & Project Structure

With objectives and research questions defined, the method of answering these questions can be set out. The two (2) research questions are fairly interwoven, and by answering one (1) question, the other question can be partially answered as well.

The question of how process mining can be used in order to optimize for resource allocation, is primarily answered by designing the process optimizer. This thesis will therefor follow a *research by design* approach. As there could be several proposals for how to implement the divide and conquer approach, these approaches have to be experimented on and evaluated. This research question will therefore be primarily approached in the design, experimentation, results and discussion section.

The question on the data requirements is partially answered by gathering the requirements during the optimizer design phase. Whilst the design phase should primarily grasp the data type requirements, other questions such as quantity and variety remain to be answered. A separate section on data setup will devoted to answering the remaining data questions.

As said above, to evaluate the decisions in the research by design track, several experiments must be set up and executed. Note that experiment datasets might be too large to feasibly fit in the appendix, so these resource will be hosted remotely, but they should remain accessible to everyone.

With the method of answering the two (2) research questions explained above, the document structure can be setup. The document will follow the following structure. First the problem statement and several preliminary definitions will be set out, and the research objectives and questions will be set out in **Introduction**, along with the project and thesis structure definition.

In the introduction a reference to existing optimization methods was already made. These existing methods, along with theories on process mining and the added value of process mining in (automated) business process management will be reviewed in **Literature review & Existing Theories**. This review serves two (2) purposes, the first being a review on the fundamental bases of this thesis, and the second being a review on how this thesis aims to make a (novel) academic contribution.

After the theoretical review has been performed, the design, and its various design options, will be presented. The design, the differing options, and the methodology will be laid out in **Method**.

The design section should provide answers to both research questions, but it is not exhaustive enough to cover all aspects of the research question related to data requirements. A full overview of the data requirements, and a review of other sources, will be provided in **Data** in order to further answer the data question.

The experiment setup will be performed in **Experiment Setup**. This section is split up in two (2) parts. The first part will cover precisely how the experiment data is to be setup, complete with parameter recordings, using a purpose-built data generator. This section will be used to develop an algorithm for the aforementioned experiment data generator, which will be instrumental to develop the experiments to test the different optimization designs proposed in this thesis. This data generator will construct data according to the data requirements found prior. The second part will be dedicated to explaining what will be experimented on and how the experiments are setup. The comparison metrics and several expectations will also be provided.

With the experiment-setup defined, experiments can be run, and results can be gathered. The results of the experiments, together with an interpretation discussion and an implication discussion, are gathered and presented in

Results.

In **Discussion** a discussion will outlie the shortfalls of this thesis and the research done, and it will provide opportunities brought forth from this thesis.

Closing off this research project and finalizing the thesis, the discussion should be concluded. The conclusion of this thesis research project is provided in the final section, **Conclusions**. The conclusion section should provide definitive answers to the two (2) research questions. The first question will be answered with qualitative and quantitative data requirements, and the second question will be answered by providing a process optimizer prototype base on the aforementioned principles. The conclusion will also offer a set of implications which follow the results from this overall research project.

2 Literature review & Existing Theories

In writing and developing this thesis, several theories on topics such as process mining, process management and datamining will be used. Before diving into the development of the optimization method which will be proposed in this thesis, it's important to understand and grasp the existing theories. This section will be devoted to introducing and reviewing the (to be) used theories in this thesis.

Besides reviewing existing theories which will be used, it's also important to understand how the proposed optimization method in this thesis will be positioned besides other optimization techniques. To that extend, other optimization methods will be reviewed and compared against the operational concepts of this thesis. Note that this thesis is not a review thesis in and of itself, so the performance of the optimizer resulting from this thesis will not be compared and tested against possible competing methods.

2.1 Existing theories on process mining

Process mining itself is not a new research field. Several well developed and studied algorithms already exist within the field of process mining. As can be seen later on in this thesis the practice of process mining has matured enough for there to be successful programs and libraries for process mining [10], [11].

Process mining the practice of constructing a process model, rather a process tree, complete with process variants, based on some set of event-data logs [1], [3]. There are several existing process mining algorithms such as for example the *α-algorithm*, which was one of the first viable process mining algorithms as it is able to successfully and satisfiably deal with process concurrency [1], [12]. Another example is the *Heuristic process miner*, which works by taking variance frequency into account and leaving out infrequent variants/occurrences in order to simplify and generalize a model [13]. A third example is the *Inductive miner*, which is regarded as one of the most forward and leading process mining and discovery methods due to its accuracy, satisfiability and flexibility [14], [15].

The inductive process miner will be the process miner of choice as it is both one of the more accurate and satisfiable miners, and it does not disregard less prevalent process variants unlike the heuristic process miner. As the process optimizer proposed in this thesis will work on a divide and conquer basis with a selected variant subset, it is important for the selection possibilities that all variants, even the smaller ones, are available.

The process optimizer of this thesis should also work regardless of the process map format or the underlying objective value structures. This is yet another reason why the inductive miner is chosen over the other methods, as the inductive miner is known to be able to model more complex process tree structures such as concurrent sub-processes. This specific process discovery algorithm will be implemented using the existing libraries provided by the PM4PY library [11].

2.2 Importance of process management and process mining

The field of process mining has been developed to get a better understanding from business and supply chain processes. This is done by analysing business generated data such as event logs. Businesses can improve their business processes using business process management techniques such as process analysis, process mining and process optimization. This is especially becoming relevant with the paradigm of big-process-data where businesses and supply chains start to record and monitor the individual activities in processes more and more [16], [17].

One element of (business) process management is allocating the required resources to activities within a process [18]. This is an important step because the mismanagement of resource allocation could lead to inefficient combinations of activities and resources in a process, and therefore an inefficient process. This thesis will propose a process optimization technique which focusses on the resource allocation

aspect of process management. This potentially makes this thesis a strong addition to the field of process management.

2.3 Current resource-activity allocation optimization methods

Resource allocation optimization is not a new concept, and several methods, frameworks and algorithms have already been developed to attempt (automated) resource allocation optimizations. This subsection will review several methods, frameworks and algorithms for process resource allocation optimizations, and compare them to the optimization strategy which will be proposed in this thesis.

2.3.1 Review: Role based allocation optimization

Each of the methods discussed bring a different approach and different novelty to the field of resource allocation optimizations. The first method worth reviewing is the method proposed by Arias Et. Al. [19]. This optimization method applies two (2) novel concepts to solve the resource allocation problem. The first interesting concept they offer is the idea of a *role of a resource*. In this role of a resource concept, the optimizer takes into account the type of activity and which role category fits most appropriately. A basic example, in for example a medical environment, would be that a nurse is most fit to provide medical care to people, but they might not be suited to repair medical devices, whereas a technician is more fit to repair medical devices, but in turn is not best fit to provide medical care. The optimizer tries to link the available roles of resource to the required roles of activities as best as it can.

Another element of this optimizer is that it does not optimize on a complete process or an activity level, but rather at a sub-process level. This sub-process level optimization concept goes hand in hand with the resource role concept, as the main idea is to determine an appropriate role at a sub-process level, instead of at an activity level, with the assumption that activities in a sub-process share the same role.

Whilst the idea of role-depended allocation is an interesting concept, it does however have one (1) major disadvantage compared to the method proposed in this thesis. This approach requires prior knowledge about the role behaviours of resources and role requirements of activities. This constraint either makes for more complex data requirements from the trace-data log in order to determine roles, or it requires a static process/model for which resources and activities are predefined and do not change. In contrast the method proposed in this thesis operates on a *zero-knowledge* basis, so it does not require knowledge about the role requirements and behaviour characteristics of activities, and role characteristics and behaviour characteristics of resources. Instead, the optimizer in this thesis is designed to extract behaviour characteristics from process traces on its own.

2.3.2 Review: Resource allocation preference-based optimization

The idea of resource role fit, or in the case of the next method, resource preferences, is an idea shared by other papers as well, as can be seen in the paper by Zhao Et. Al. [20]. In this paper resource preference, and the related performance to resource preference, are used as one of the main allocation constraints. The idea is that a resource can perform different activities with differing preferences and with different levels of efficiency. However, this again either complicates requirements for the datasets where preference metrics must be recorded in one way or another, or it requires a static model where resource preferences are pre-determined. This paper relates resource preference to resource performance, but in a way, this is also happens dynamically and again with a *zero-knowledge* basis in the method proposed in this thesis. The method in this thesis constructs a performance model, later named a cost model, for each resource-activity combination. If performance is then related to preferences, then the method in this thesis essentially extracts preferences by simply looking at the best and worst performing discovered cost models.

2.3.3 Review: Reinforcement learning based Markov decision process optimization

The a-priori resource or activity knowledge approach seems to be a common trend in several methods, as it is again repeated in the paper by Huang Et. Al. [21]. This paper also uses resource on activity behaviour as one of its key aspects, but rather than developing models for each resource, it uses reinforcement learning to determine a best fit. Something else this paper brings up is the optimization level depth. Where the previous two (2) papers optimize on a sub-process level and an activity level, this paper optimizes on a process level. This is similar to the concept proposed in this thesis, as it is argued that processes are interdependent on their own activities and sub-processes, meaning, optimizing one part of a process might improve the performance for that part, but for the entire process it could decrease the performance. However, where this paper differs with the method proposed in this thesis is in how they approach whole process optimization. This paper approaches whole process optimization by converting a process into a *Markov Decision Process* where the process is converted into a mathematical stochastic model with component level probabilities. This stochastic complete model is then used to perform probability friendly optimizations. This thesis proposes a divided and conquer approach where a process is pruned into simpler deterministic process-variants (so without any probabilities) which are then optimized, and the optimization results are merged. Each method has their own benefits and drawbacks.

The Markov Decision Process conversion method has as its two (2) main benefits that only one (1) optimization run has to be performed as there is only one (1) model. The second benefit is that there are no separate steps required to merge optimization results into a final result. However, the main drawback is that the stochastic model is a more complex model to construct, to model, and to optimize for.

The divide and conquer approach has the primary benefit that the process variants are simpler to model for and easier to optimize for. When pruning the process into simpler variants, variants can also be left out if they are deemed unnecessary for the primary business goals. The main drawback is the need to perform multiple optimizations, as there are multiple models, and to merge these optimization outcomes.

Another difference between the method proposed in this thesis, and the methods found in the other papers, is the use of less flexible, but simpler linear models, as opposed to using polynomial models resource behaviour models. One example of a method similar in optimization approach is the method proposed by Korhonen P. and Syrjänen M. [22]. This paper attempts to convert its problem in a multi-objective linear programming problem. It is similar in that it attempts to convert the process optimization problem in a mathematical programming problem, just like the method in this thesis will do. But where it, and other papers as well, differ from this thesis is the exclusion of the, in theory, more flexible polynomial modelling approach.

2.3.4 Knowledge gap

To summarize, there are several methods already existing for resource allocation strategies, however there are a few large differences between these known methods and the method proposed in this thesis. The resource allocation method in this thesis will serve as a complete and expandable framework, where resource and activity behaviour and cost models are discovered using available data. The other methods use a more a-priori approach where the behaviour of resource and needs of activities have to be known beforehand. This thesis proposes a divide and conquer approach for whole process resource allocation optimization, where other methods optimize on an activity, or sub-process level, or use a stochastic modelling approach. And finally, the last major difference, the method proposed in this thesis should also support polynomial cost and behaviour modelling, as compared to other methods which are limited to linear modelling in order to simplify the optimization step. From this, three (3) gaps in previous knowledge can be defined which this thesis aims to fill:

- **DIVIDE AND CONQUER WHOLE PROCESS OPTIMIZATION** Other methods focus on activity level, sub process level optimization as opposed to whole process optimization, or they optimize for the entire process using a single complex stochastic model instead of using multiple smaller simpler deterministic models in a divide and conquer approach;
- **DYNAMIC COST MODEL DEVELOPMENT** Instead of relying on an a-priori approach of predefining cost models, this thesis will explore whether model can be dynamically generated and then applied to an optimization model;
- **POLYNOMIAL COST MODEL APPLICATION** Existing research seems to mostly rely on linear models instead of more flexible polynomial models for resource allocation optimization. This thesis will explore whether polynomial cost models are feasible for application.

The first two (2) gaps in knowledge serve a primary focus in this thesis, whereas the last knowledge gap is an extension to the dynamic cost model component.

3 Method & Design

The introduction in **Introduction** of this thesis that the process optimizer will focuss on resource allocation optimizations. This optimizer works on the principles of process mining where an event-log is analysed to create insight in a business. However, in this thesis, a design will be proposed to add an additional insight besides process visualization: an optimal combination of resource-activity allocations, based on event data, using a divide and conquer optimization approach with a flexible cost model setup.

In the introduction, specifically in **Problem Statement & Project Definition**, several important definitions for this project have been laid out, which will be leading for the design of the optimizer. First, the optimizer will focus on optimizing the following objectives:

- **THROUGHPUT TIME** how long, in terms of time, does a step in a process, or the entire process take, to complete;
- **WAIT TIME** how much time exists between the *end* and *start* of steps in a process;
- **COST** how much does it cost, in terms of monetary amounts, to execute a step in a process, or the entire process.

Secondly, the optimizer will be **NON-INTRUSIVE**, meaning it does not change a given process, but rather change the parameters, in this case the allocation of resources within this process, of a given process. It will do this in a divide and conquer style by simplifying a (stochastic) (business/supply-chain) process into multiple simpler deterministic processes. These two (2) constraints and objectives will be leading in the design and development of the optimization method.

The full implementation of the prototype based on the described algorithm in this section will be made available in a repository related to this thesis. This repository is accessible through *"https://github.com/MarcelKolen/process-mining-resource-allocation-optimizer"*.

3.1 Optimization routine

The resource-activity allocation optimizer will follow a set of steps, or rather sub-routines, to perform resource allocation optimization. These steps as follows:

1. **DATA INGESTING** a given dataset, of varying file format, is imported into the optimizer;
2. **DATA LABELLING** key columns of the dataset are labelled according to the data requirements;
3. **PROCESS TREE GENERATOR** from a given dataset, construct a process tree representing the process which created the dataset using process mining tactics;
4. **DETERMINE PROCESS VARIANTS** a process might consist of several variants which should be distinguished from one and another. These variants play a key role in the divide and conquer approach of this optimization method;
5. **DETERMINE LEGAL RESOURCE-ACTIVITY ALLOCATIONS** within a process, only a limited set of combinations of activities and resources exist, the optimizer should find all legal combinations;
6. **FIND RESOURCE COST FIGURES** for every legal resource-activity combination, a cost model, in both wait time and throughput time, and monetary values, should be constructed;
7. **PRUNE PROCESS TREES** for every variant of the found process, construct a pruned process tree, representing the variant in a simplified and deterministic process tree, based on the original process tree;
8. **PERFORM VARIANT OPTIMIZATIONS** for every variant, run the optimization algorithm finding the optimum combination of resources and activities for that particular variant;
9. **MERGE VARIANT OPTIMIZATIONS** every variant of the process has an optimal version of the allocations. These versions must be merged into a global optimum allocation set, optionally taking into regard variant precedence. This is the closing step of the divide and conquer approach.

Every step in the routine above will be further elaborated on in more detail in the following (sub)sections.

3.1.1 Data ingesting

The step of data ingesting converts the input data, from a variety of possible input files into a format which is understandable by the optimizer. The format used within the prototype for this thesis will be a Pandas DataFrame from the Pandas Python library [23], [24]. The input files could be among the following, but in theory not limited to, common file formats: .CSV, .JSON, .XML, .XES, .XLS, .XLSX, .XLSM, .XLSB.

3.1.2 Data labelling

In the data labelling step, the optimizer is told by the user which columns contain what specific data. The user will indicate to the optimizer which column corresponds to which data-type. Working with the three (3) objective functions, this thesis defines the data columns requirements as follows:

- **CASE IDENTIFIER**;
- **ACTIVITY IDENTIFIERS**;
- **START TIMESTAMP**;
- **RESOURCE IDENTIFIERS**;
- **ACTIVITY EXECUTION DURATION TIME**;
- **ACTIVITY WAIT TIME**;
- *(Optional)* **END TIMESTAMP**;
- **ACTIVITY EXECUTION COST**.

3.1.3 Process Tree Generator

Using the labelled data, the optimizer should be able to construct a process tree using the concepts of process mining. The prototype for this thesis will utilize the PM4PY Python library [10], [11], which contains several process mining routines build in.

The PM4PY Python library offers several process discovery methods. These methods have been reviewed, compared and discussed in **Existing theories on process mining**. As was stated in that review, the *inductive process miner* seems to be the best fitting process discovery algorithm for the optimization method covered in this thesis.

The discovery and construction of a process tree from process data is required because the optimizer will later use the process discoveries to construct pruned process tree variants, which are in turn used for optimizations in a divide and conquer style approach.

3.1.4 Determine process variants

As stated earlier in the routine description, the process tree is going to be pruned into a set of process variants as part of its divide and conquer approach. The pruning step is performed to simplify the complex, and stochastic process model into multiple smaller, simplified, and deterministic models for the optimization step. In order to perform the pruning step, the different variants of a process need to be extracted from the process data log.

As with the process tree generation step, the finding of process variants will be performed by build in functionalities of the PM4PY Python library [10], [11]. This particular functionality in the PM4PY library which will be used for the prototype, gives both a list of different variants, and for every variant an occurrence count.

The occurrence count for every variant is an important metric to record. If a variant occurs more often, than others, it could indicate that this variant is more important to optimize for. Further on in this thesis,

different merging techniques for variant optimums are discussed in **Merge variant optimizations**. Here, in the different merging techniques, the occurrence count of a variant is used to order variants by precedence.

Note that a variant should be determined by the activities executed in a process, and their respective order in time. A process could for example have:

- **PARALLEL SUB-PROCESSES** these are sub-processes for which multiple branches could run concurrently, but do not necessarily have to start at the same time. However all branches do have to finish before the next activity after the parallel sub-process can be started;
- **XOR- SUB-PROCESSES** which are sub-processes where for a particular process case, only one of the available (two (2) or more) sub-processes may be executed;
- **XOR-LOOPS** in which the first element of a loop is always executed at least once, but the other components of the loop do not have to be executed. However, if the other components, or rather activities of an XOR-loop are executed, the first element must also be executed once more. Note that both the first element and the looping component can both be larger sub-processes consisting of multiple activities.

Taking these components into regard, a variant is then determined by the particular branches it follows in an XOR-sub-process, and how often it performs a loop. It is however not determined by the order in which it executes its parallel branches.

To help visualize the description of variant-distinction above, consider the following process.

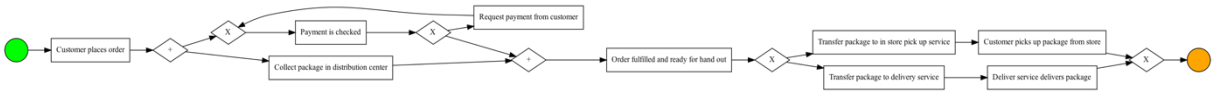


*Figure 2 – Process model example 1, represented in a BPMN schema. Please refer to **10.2** for a larger version.*

This process contains a parallel component, an XOR-loop, and at the end an XOR component. In this specific process, the differing variants can occur in either the XOR-loop and/or the XOR-sub-process. Consider the following three process logs to see the different distinguishes between variants.

Table 1 – Process Model Example 1, Trace 0

Activity	Resource	Cost	Start Timestamp	End Timestamp
Customer places order	-	0	17:00:00	17:00:10
Collect package in distribution center	Jim	82	17:00:10	17:09:32
Payment is checked	Erik	11	17:00:10	17:02:22
Order fulfilled and ready for hand out	Ingrid	27	17:09:32	17:13:23
Transfer package to in store pick up service	Jim	20,5	17:13:23	17:15:43
Customer picks up package from store	-	10	17:15:43	17:35:43

Table 2 – Process Model Example 1, Trace 1

Activity	Resource	Cost	Start Timestamp	End Timestamp
Customer places order	-	0	16:00:00	16:00:10
Payment is checked	Shenna	9.5	16:00:10	16:01:58
Collect package in distribution center	Jim	82	16:00:10	16:09:32
Order fulfilled and ready for hand out	Ingrid	27	16:09:32	16:13:23
Transfer package to in store pick up service	Jim	20.5	16:13:23	16:15:43
Customer picks up package from store	-	10	16:15:43	16:35:43

Table 3 – Process Model Example 1, Trace 2

Activity	Resource	Cost	Start Timestamp	End Timestamp
Customer places order	-	0	08:00:00	08:00:10
Payment is checked	Shenna	9.5	08:00:10	08:01:58
Collect package in distribution center	Jim	90	08:00:10	08:08:34
Request payment from customer	Erik	55	08:01:58	08:12:58
Payment is checked	Shenna	9.5	08:12:58	08:14:46
Order fulfilled and ready for hand out	Pete	33	08:14:46	08:17:55
Transfer package to delivery service	Tim	22.5	08:17:55	08:21:20
Deliver service delivers package	Tim	60	08:21:20	08:44:08

The three (3) tables above represent two (2) variants. The first variant is represented by both ***Table 1*** and ***Table 2***. Note that rows **TWO** (2) and **THREE** (3) (with a row count starting from one (1) as the first row) between ***Table 1*** and ***Table 2*** are different. The two (2) rows are swapped between the two (2) process traces. However, as indicated by ***Figure 2***, these two (2) activities are executed in parallel. Even though the order in time is different, for the definition of variants in this thesis, they are still the same variant because order in time is not taken into regard with parallel branches.

However, when comparing ***Table 3*** to ***Table 1*** and ***Table 2***, a clear difference in variants can be observed. First, the XOR-loop in the process is initiated, and a different XOR-branch is picked. Either one of these two (2) differences would constitute a different variant, this trace happens to show both differences.

As a **FINAL IMPORTANT NOTE** on finding the different process variants: a process tree might have more variant options than is extractable from the available event data. However, in this thesis, and for the optimizer developed in this thesis, only the variants exhibited in the event data are taken into regards.

3.1.5 Determine legal resource-activity allocations

Determining which resource-activity allocation combinations are legal will be vital for constructing the process models and running optimizations. In this thesis, the legality of a resource-activity allocation is simply determined by analysing which combinations occur in the event data.

For every distinct resource which is found in the event data, all activities to which they were allocated have to be found. Only the found combinations will be considered legal. Given an event data set, it could occur that certain resources are able to perform multiple activities. At the same time, an activity might be executable by multiple resources. These two (2) possibilities allow for the construction of a **RESOURCE-ACTIVITY ALLOCATION MAP**. This resource-activity allocation map, or subsets of it, can be seen as a preliminary to the *input options* the optimizer has.

Besides legal combinations of resources and activities, another factor needs to be taken into regard. A scenario could arise where a resource might only be allocated a certain number of times. This constraint will be determined by counting the number allocation occurrences in all cases and taking the maximum value. The maximum allocation is not a summation of all occurrences in all cases. The exact determination for the resource allocation capacity is determined as follows:

$$\forall r \in R; c_0, c_1, \dots, c_{|C|-1} \in C: r_{ma} = \text{Max}\{\text{O}(r, c_0), \text{O}(r, c_1), \dots, \text{O}(r, c_{n-1}), \text{O}(r, c_n)\}$$

Equation 1 – Where R is the set of all resources and r is a resource from that set, C ($C = \{c_0, c_1, \dots, c_n\}$) is the set of all process cases and c_i is a case from that set, r_{ma} is the maximum allocation allowance for a particular resource, Max *takes the maximum value of its set of inputs, and* $\text{O}(x, y)$ *find the occurrence of x in y.*

An important note to take into consideration is that in this thesis, no special attention will be given to parallel resource allocation restrictions. To simplify the prototype, the prototype which will be developed for this thesis will allow parallel allocations, as long as it abides to the above-described restrictions of legal allocation combinations and allocation limitations.

In order to provide a visualisation of the expected outcome of the resource-activity allocation map, consider the traces in ***Table 1***, ***Table 2*** and ***Table 3***. When applying the methodologies described above, the following resource-activity allocation map can be constructed as shown in ***Table 4***.

Table 4 – Process Model Example 1, Resource-activity allocation map from traces 0, 1, 2

	Activity	Resource
0	Customer places order	-
1	Collect package in distribution center	Jim
2	Payment is checked	Erik
3	Payment is checked	Shenna
4	Request payment from customer	Erik
5	Order fulfilled and ready for hand out	Ingrid
6	Order fulfilled and ready for hand out	Pete
7	Transfer package to in store pick up service	Jim
8	Customer picks up package from store	-
9	Transfer package to delivery service	Tim
10	Deliver service delivers package	Tim

Accompanying the resource-activity allocation map, should be an allocation limit table, which indicates for every resource how often it may be allocated:

Table 5 – Process Model Example 1, Resource allocation limit from traces 0, 1, 2

Resource	Allocation limit
Jim	2
Erik	1
Shenna	2
Ingrid	1
Pete	1
Tim	2
-	2

As can be observed between ***Table 4*** and ***Table 5***, some resources, or employees in this example, such as Erik, might be allocatable to more than one (1) activity, but they can only be allocated once. Other resources, such as Shenna, can be allocated multiple times, but they are only assignable to one (1) activity.

3.1.6 Find resource cost figures

In order to perform the optimizations on the objectives defined in **Optimization definition**, the cost of allocation for every resource-activity combination needs to be determined. The cost of allocation in this thesis will be captured in both monetary cost, and time related costs such as wait time and throughput time. This cost will be translated into a model which plays and instrumental part in the modelling of a process.

In this thesis, the theory is used that the allocation of a resource to two (2) or more activities affects its cost behaviour in any of its objective functions. This theory originates form the inverted U-theory [25]–[27], which is further elaborated on in **Resource simulation values**. An additional assumption which is made is that an activity occurring multiple times will also affect its cost behaviour. For example, one such scenario could be imagined as follows, if an employee is allocated twice in a process, the allocation in monetary terms could become cheaper, however, they could take longer to perform the activities. A second scenario would be the following, say an activity is involved with quality control and repairs. This activity occurs when there is a problem with a product or service. If this activity reappears after the initial attempt of solving this issue, then a scenario could exist where the problem is more complex than initially thought of, so this next attempt of solving the issue will require more time and effort. This assumption could affect the way the optimizer chooses to pick its optimal resource-activities allocation combinations.

Taking the above theories into regard, the way the cost for an allocation is determined adds some complexity to the overall process. Instead of finding the associated cost values, in time and monetary regards, now the optimizer needs to define functions of cost for at least two (2) variables. These functions of cost could be determined through multi-variate regression over the available event data. Because there are multiple methods for approaching this issue, a separate section is introduced in **Cost Figure Variants** where this issue is elaborated on in more detail.

The functions of cost should be functions with the following parameters and variables:

- **RESOURCE** (r) the resource which is going to be allocated to an activity;
- **ACTIVITY** (a) the activity to which the resource is allocated to;
- **ALLOCATION COUNT** $\big(O(r,s)\big)$ the number of allocations of this particular resource, where s is the simulated optimized (variant) case;
- **ACTIVITY OCCURRENCE COUNT** $\big(O(a,s)\big)$ the number of allocations of this particular activity, where s is the simulated optimized (variant) case.

The parameters and variables above should provide a simple baseline for inputs to construct cost functions. However, just like with regular datamining exercises, more parameters, or inputs, could be used. This makes the models more complex, but it also adds a greater degree of flexibility and insight.

3.1.7 Prune process trees

To implement the divide and conquer approach a pruned process trees for every variant will be developed. In this pruned variant, all XOR-sub-processes are resolved to be one (1) sequential branch, and all loops are unrolled into sequential branches. This means that only the relevant branch in the XOR-sub-processes will remain, and that for the loop, all repeated elements are put in sequence after one and another. Pruning of XOR-sub-processes and loops is done to remove probabilistic elements and convert the process from one (1) large stochastic model into multiple smaller, deterministic models.

A structure which will remain in pruned process trees, are parallel sub-processes. This has two (2) reasons. In a parallel sub-process, all branches are executed concurrently, so there is no branch to cut out. If there are XOR-sub-processes or XOR-loops within branches of a parallel sub-process, these components will be pruned and resolved to simplify the individual parallel branches. The second reason is because a parallel branch carries significance in throughput time. As stated before in **Determine process variants**, branches in a parallel branch do not have to start concurrently, but the next activity can only start when all branches have finished. This means that the throughput time of a parallel branch is never shorter than its longest lasting branch.

In order to visualize the pruning concept, consider the process traces, and two variants, as presented in **Determine process variants**, in ***Table 1***, ***Table 3***. The original process tree, as presented in ***Figure 2***, is represented in a tuple-list structure below.

```
('>',
 ['Customer places order',
  ('+',
   ['Collect package in distribution center',
    ('*', ['Payment is checked', 'Request payment from customer'])]),
  'Order fulfilled and ready for hand out',
  ('X',
   [('>',
    ['Transfer package to delivery service',
     'Deliver service delivers package']),
    ('>',
     ['Transfer package to in store pick up service',
      'Customer
```

Figure 3 – Process model example 1, represented in a textual schema

This representation knows four (4) different operators. These operators are defined as follows:

- **> A SEQUENCE OPERATOR** all children of this operator are executed in a sequence of one after another (Left to right, top to bottom).;
- **+ A PARALLEL OPERATOR** all children of this operator are executed concurrently. They may start at the same time, and the element after the parallel operator can only start once all children of the parallel branch are concluded;
- *** AN XOR-LOOP OPERATOR** has two (2) children, of which the left most (or topmost) element is always executed, and the other child is optionally executed after the first element. If the optional child is executed, the first child must be executed (again) as well;
- **X AN XOR-BRANCH OPERATOR** has two (2) or more children, of which only one (1) may be executed per instance of a process.

Now consider the variant represented in ***Table 1***. This variant does not initiate the payment loop, and it opts for the second of the two (2) XOR-sub-process branches. Its process tree is represented in a tuple-list style, in ***Figure 4***, below.

```
('>',
 ['Customer places order',
  ('+',
   ['Collect package in distribution center',
    'Payment is checked']),
  'Order fulfilled and ready for hand out',
  'Transfer package to in store pick up service',
  'Customer picks up package from store'])
```

*Figure 4 – Process model example 1, variant representing **Table 1** and **Table 2***

As stated in **Determine process variants**, this particular variant also covers the trace in ***Table 2*** even though the trace is slightly different.

Finally consider the variant in ***Table 3***. This variant does initiate the payment loop, and it opts for the first of the two (2) XOR-sub-process branches. Its process tree is represented in a tuple-list style, in ***Figure 5***, below.

```
('>',
 ['Customer places order',
  ('+',
   ['Collect package in distribution center',
    ('>',
     ['Payment is checked',
      'Request payment from customer',
      'Payment is checked'])]),
  'Order fulfilled and ready for hand out',
  'Transfer package to in store pick up service',
  'Customer picks up package from store'])
```

*Figure 5 – Process model example 1, variant representing **Table 3***

3.1.8 Perform variant optimizations

With the pruning of process trees, for the different variants, completed, the optimization step can be executed. The optimization step will be performed on the pruned trees of the different variants as the divide and conquer approach of this optimizer.

In **Determine process variants** the concept of determining variants is introduced and the importance of occurrence recording is discussed. This variant optimization concept is based on the Pareto principle which, when extended to processes, provides the following (paraphrased) idea: *"20% of a business'*

processes/supply line structures generates 80% of the overall profits" [28]. In **Merge variant optimizations techniques** the *Pareto 80/20* principle is applied to a certain degree when merging different variants optimizations into one (1) complete optimized process. In this thesis, *"a process"*, as stated in the Pareto principle, is a variant of the overall business process. In In **Variants to optimize selections** the concept of selecting only a subset of process-variants is discussed.

On to the actual discussion of the optimization step: In this step, an optimized resource-activity allocation combination is determined for every variant of the selected subset of variants. In order to perform the optimization step, four (4) components are required:

- **OBJECTIVE FUNCTIONS** the function for which the optimizer will find a most optimal value, which is the smallest value in this particular problem setting;
- **INPUT VARIABLES** the variables which the optimizer can tweak and adjust in order to manipulate the outcome of the objective function;
- **CONSTRAINTS** the borders and rules within which the optimizer needs to operate, and to which the optimizer needs to adhere;
- **PROBLEM TYPE** the problem type determines how the optimization problem needs to approached from a formal constraint definition and input variable type standpoint.

3.1.8.1 Objective Functions

This section will develop the (three (3)) objective functions into mathematical definitions used for the optimization steps. Since the optimization problem in this thesis has multiple objective functions, a separate section on optimizing for multiple objectives at the same time has been introduced in **Multi-objective optimization**.

MONETARY COST The first objective is the *monetary cost* objective. Within this thesis, this objective function is a simple objective, as it's, in its fundamental basis, a completely linear function. To determine the cost of a (variant of a) process, all costs of executing activities with certain resources need to be simply summated. Parallel sub-processes, as defined in **Determine process variants** do not require special treatment. These branches operate concurrently, and their respective cost is the sum of the costs of in these branches. With the above in mind, the objective function for *monetary cost* can be defined as follows:

$$r_0, r_1, \dots, r_{|A_v|-1} \in \overline{R_v} : o_c(r_0, r_1, \dots, r_{|A_v|}) = \sum_{i=0}^{|A_v|} \gamma(r_i, a_i)$$

*Equation 2 – Where A_v is the set of all activities of a particular variant v, and 'a' is an activity in this set, $\overline{R_v}$ is the set of all resources which may be allocated to activities in this variant v and r_ρ is a resource from within this set specifically allocated to an activity, o_c is the result of the objective function for costs, and $\gamma(x, y)$ is the cost of executing activity 'y' with resource 'x'. The cost of executing an activity is a result of a regression function which uses historical data for a particular resource and activity allocation combination, specifically looking at the effects of a resource being allocated more than once. $\gamma(x, y)$ can be substituted for one of the proposed models (multivariate linear or polynomial) from **Cost Figure Variants**, specifically setup for the monetary cost component. o_c takes an input as a list of resources which are allocated to activities. Note that two resource inputs, say r_0 and r_1 may point to the same resource.*

THROUGHPUT TIME Next is the *throughput time* objective. The throughput time objective is similar to the monetary cost objective, as it is a sum of its time components, however, one (1) deviation is that *parallel branches* are not approached as a simple sum. In a parallel branch instance two (2) or more branches may start at the same time, they can occur concurrently, and the next activity after the branches may only start once the last/latest branch has finished executing. For the sake of simplicity during the optimizations, wait-time is not taken into consideration so that only run time dictates the slowest branch. This simplifies the objective in that only the longest branch time needs to be considered for the summation. This means parallel components can be modelled using *Max* expressions. The objective function of *throughput time* is defined as follows:

$$\forall r_0, r_1, \dots, r_{|A_v|-1} \in \overline{R_v}: o_t\left(r_0, r_1, \dots, r_{|T|-1}; T_v\right) =$$

$$\Theta\left(r_0, r_1, \dots, r_n; A_v - P_v(T_v)\right) +$$

$$\sum_{P \in P_v(T_v) | p_0, p_1, \dots, p_{|P|} \in P} \text{Max}\begin{Bmatrix} o_t\left(r_0, r_1, \dots, r_{|p_0|}; p_0\right), o_t\left(r_0, r_1, \dots, r_{|p_1|}; p_1\right), \dots, \\ o_t\left(r_0, r_1, \dots, r_{|p_{|P|-2}|}; p_{|P|-2}\right), o_t\left(r_0, r_1, \dots, r_{|p_{|P|-1}|}; p_{|P|-1}\right) \end{Bmatrix}$$

Equation 3 – Where T_v is a process tree of variant v, $\overline{R_v}$ is the set of all resources which may be allocated to activities in this variant v and r_ρ is a resource from within this set specifically allocated to an activity. $P_v(x)$ is the set of all parallel tree occurrences for a given process-(sub)tree x, and P is a parallel subtree which itself is a set of two (2) or sub trees (also called branches of in a parallel tree). p_c is a branch of a parallel component P where is c is the count of a branch. $\Theta(R_i; A_i)$ is the function described in **Equation 4***, where R_i is the set of all resources allocated to the activities from A_i, and $A_v - P_v(x)$ (see* **Figure 6***) is the remaining set of activities after subtracting all activities in any and all parallel branches. This function is defined recursively to deal with nested parallel components. Finally,* Max *takes the maximum value of its set of inputs, and o_t is the result of the objective function for throughput time.*

$$r_0, r_1, \dots, r_{|A|} \in \overline{R_v}: \Theta\left(r_0, r_1, \dots, r_{|A|-1}; A\right) = \sum_{i=0}^{|A|} \theta(r_i, \bar{a}_i)$$

Equation 4 – Where A_v is the set of all activities of a particular variant v, and 'a' is an activity in this set, $\overline{R_v}$ is the set of all resources which may be allocated to activities in this variant v and r_ρ is a resource from within this set specifically allocated to an activity, $\Theta(R, A)$ is the result of the objective function for throughput time of a sequential set of activities, and $\theta(x, y)$ is the time required of executing activity 'y' with resource 'x'. The time required is a result of a regression function which uses historical data for a particular resource and activity allocation combination, specifically looking at the effects of a resource being allocated more than once. $\theta(x, y)$ can be substituted for one of the proposed models (multivariate linear or polynomial) from ***Cost Figure Variants****, specifically setup for the throughput time component.*

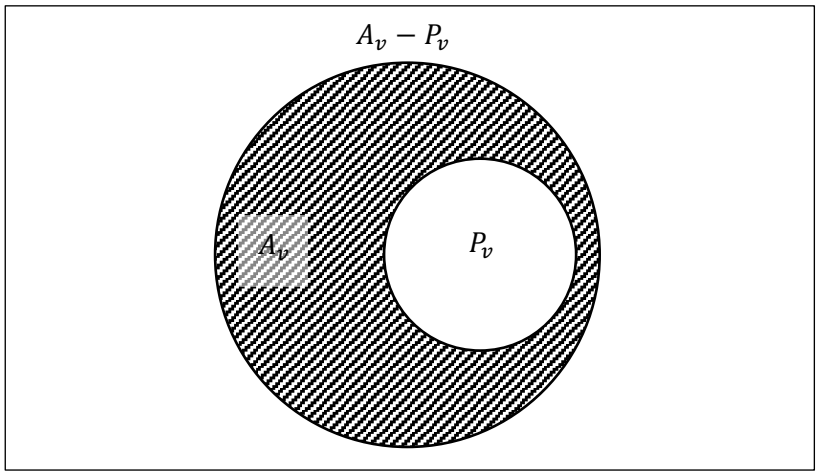


Figure 6 – $A_v - P_v$ which indicates only the diagonally hatched section of the set A_v. (Note that P_v is a subset of A_v)

WAIT TIME Lastly, the *wait time objective* can be defined. The wait time objective will only be developed conceptually in this thesis, it is not included in the prototype. The wait time in this thesis is defined as the total wait time over all activities. Just as with the cost objective, parallel sub-processes are not given any special treatment, as the total wait time over all activities is to be regarded. With the above in mind, the wait time objective can be defined as follows:

$$r_0, r_1, \dots, r_{|A_v|-1} \in \overline{R_v}: o_w(r_0, r_1, \dots, r_{|A_v|-1}) = \sum_{i=0}^{|A_v|} \omega(r_i, a_i)$$

*Equation 5 – Where A_v is the set of all activities of a particular variant v, and 'a' is an activity in this set, $\overline{R_v}$ is the set of all resources which may be allocated to activities in this variant v and r_ρ is a resource from within this set specifically allocated to an activity, o_w is the result of the objective function for costs, and $\omega(x, y)$ is the sum of half of the wait time before and after executing activity 'y' with resource 'x'. The wait time before and after executing an activity is a result of a regression function which uses historical data for a particular resource and activity allocation combination, specifically looking at the effects of a resource being allocated more than once. $\omega(x, y)$ can be substituted for one of the proposed models (multivariate linear or polynomial) from **Cost Figure Variants**, specifically setup for the wait time component.*

3.1.8.2 Input variables

In the development of the conceptual objective functions in **Objective Functions**, two (2) input variables have already been introduced. These two (2) input variables are also the only input variables that need to be considered in order to perform optimizations. The input variables are *resources* and *activities*, or rather more specifically, the combination of resources and activities.

In order to simplify the input variables, the two (2) variables, resources and activities, will be combined into one (1) variable in the form of an input array with a single dimension. This variable will be called the *input allocation table*, and it consists of all legal combinations of resources and activities for a certain variant. This allocation table is a binary table where every *true*, or one (1), value indicates that a resource is allocated to a certain activity, and every false, or *zero* (0), value indicates that it is not allocated.

The input allocation table is different for every variant, as it only needs to contain the activities (and respective resources) which occur in a variant. Take the trace from ***Table 3***, its corresponding variant (as indicated in ***Figure 5***), and the original resource-activity table in ***Table 4***. The input allocation table for this variant would be the like the following table:

Table 6 – Input allocation table for the variant associated with the trace in **Table 3**. *Where ι signifies the input possibilities from a non-empty binary set.*

Activity	Resource	Allocated $\iota \in$ {true,false}
Customer places order	-	
Collect package in distribution center	Jim	
Payment is checked	Erik	
Payment is checked	Shenna	
Request payment from customer	Erik	
Order fulfilled and ready for hand out	Ingrid	
Order fulfilled and ready for hand out	Pete	
Transfer package to delivery service	Tim	
Deliver service delivers package	Tim	

3.1.8.3 Constraints

The last base-component for setting up the optimization step is the set of constraints. This optimization problem, as developed in this thesis, only knows two (2) constraints within which the optimizer must operate. Note that these are only theoretical constraints based on the model constructed in this thesis. This model could be extended with one (1) or more constraints to consider extra dimensions of process/supply chain bottlenecks.

The first constraint is as follows: Every activity must have at least and at most one (1) resource allocated to it. This stems from the idea that an activity can only be executed if there is a resource allocated to it, and that only one (1) resource at the same time can perform an activity. However, there are scenarios thinkable where an activity might require multiple resources in order to function. There are also occasions where an activity is repeated in a process multiple times, and in its repeat a different resource might be allocated to each separate occurrence. But, for simplicity's sake, the model in this thesis, and the prototype based on this thesis' model, will only consider one (1) resource per activity and no more.

The constraint will be defined as being an equality constraint, where the constraint function must equal zero (0) in order to be satisfied. With this and the above in mind, the constraint can be defined as follows:

$$\forall r_a \in R_v, \forall \bar{r}_a \in R_v : 0 = \sum_{i=0}^{|A_v|} \mathrm{SC}\left(r_{a_i}, \bar{r}_{a_i}, a_i\right) - |A_v|$$

Equation 6 – Where A_v is the set of all activities of a particular variant, and 'a' is an activity in this set, R_v is the set of all resources for a variant v, r_a is the subset of resources from within this set which may be allocated to an activity, and $r_{\bar{a}}$ is the subset of resources from within this set specifically allocated to an activity. $SC(x, y, z)$ is the single-count over activity 'z' with resources 'x' and allocated resource 'y', where it will provide a value of one (1) if one (1) and only one (1) resource (in 'y') is allocated to the activity 'z' (with the condition that $y \in x$) and it will provide a value of zero (0) if zero (1) or more than one (1) resource (in 'y') is allocated to the activity 'z' (or if $y \notin x$).

This constraint works on the following basis: the $-|A_v|$ component is the size (length) of the activities set for a variant. If there are ten (10) activities in this set, then its size is also ten (10). Because this is the difference over the sum of the XOR components, the constraint can only be zero (0), therefor satisfied, if for all activities, there is only one (1) resource allocated.

The second constraint has to be defined to ensure that the allocation limit, as described at the end of **Determine legal resource-activity allocations**, is not overshot. As a short recap: every resource will have a limited allocation budget (so how often they can be allocated to activities in a process).

With the above description in mind, and with the notion that this constraint will also be an equality constraint, the resource allocation limit constraint can be defined as follows:

$$0 = \sum_{i=0}^{|R|} \mathrm{Max}\left\{\mathrm{C}_r(r_i), b_{r_i}\right\} - \sum_{i=0}^{|B|} b_i$$

Equation 7 – Where R is the set of all resources, r is a resource from this set. B is the set of resource allocation budgets, and b_i is the allocation budget of a resource, and b_{r_i} is the specific allocation budget of resource r_i. Max *takes the maximum value of its set of inputs, and $\mathrm{C}_r(x)$ finds the number of total occurrences of the given resource x.*

This constraint works on the following basis: first the sum of all budgets for all resources is calculated. This sum is then subtracted from the Max sum component. The Max sum component should equal the total budget if and only if for every resource the total number of allocations is smaller or equal to its budget. If one (1) resource is above the budget, the max sum component exceeds the sum of all budgets, making the constraint inequal thereby failing the constraint.

An issue with the second constraint is that its bases its validity solely on historical data. The maximum budgets of the resources are based on what the maximum recorded allocation budget is in all data records as is described in **Determine legal resource-activity allocations**. However, in real world applications, this maximum recorded limit might be because of a "freak incident" where certain resources had to be pushed to meet process demands. This is however not entirely problematic, as the constraint simply relies on an allocation budget table. A method to prevent unwanted over allocation, as result of "freak incident" recording, is to pre-process the allocation budget table by enforcing an artificial allocation limit, where for each row the allocation budget is set to: $\mathrm{Min}(b, l)$ where Min takes the smallest value of its inputs, b is the recorded budget, and l is an artificially enforced budget.

3.1.8.4 Problem Type

With the three (3) base-components defined, and the problem statement defined, it's important to define the problem type. The problem type determines the approach with which the optimization of a problem-space is attempted. Within the world of optimization problems, there exist many different forms of optimization settings and problems. Examples range from Convex Linear Programming, Integer Linear Programming, and Integer Non-Linear Programming, to Mixed (Integer) Linear Programming and Mixed (Integer) Linear Programming [29], [30]. In order to define the overall problem type, all

components, being the objective functions, the input variables, and the constraints, have to be broken down into a type. Ideally, if possible, the components should be converted or formulated as linear integer problem types in order to simplify the optimization as they will then be NP-complete as compared to NP-hard [30]. A more in-depth reasoning for the advantage of the optimization being integer linear as compared to other types is given at the end of this sub-section.

First the **INPUT VARIABLES**. The input variables, as defined in **Input variables**, are linear variables, or more specifically, 0 – 1 linear integer variables in the form of binary variables. The input component of this optimization problem is therefor a linear problem component.

Second are **THE CONSTRAINTS**. The constraints of this problem are defined in **Constraints**, and are essentially linear limitations to the input variables. In their definitions, they appear to be non-linear, due to the use of the single-count component, and Max component, but this is only the case for the collectivised constraints. The constraints described earlier, can be seen as a combined constraint of many smaller constraints. Each of these smaller constraints can be described as a linear constraint of the linear input variables.

The first constraint described in ***Equation 6***, could for example be broken down per activity. Take for example the activity *Order fulfilled and ready for hand out* from the input table in ***Table 6***. For this activity the constraint can be redefined as an equality constraint (only satisfies when the constraint evaluates to zero (0)) as such:

$$r_5 \in \{0,1\}, r_6 \in \{0,1\}: 0 = 1 \cdot r_5 + 1 \cdot r_6 - 1$$

Equation 8 – Where r_5 signifies whether Ingrid is allocated (value of one (1)) or not (value of zero (0)), and r_6 signifies whether Pete is allocated (value of one (1)) or not (value of zero (0)).

A more general description of this per-activity $a \in A$ constraint can be described as follows:

$$0 = \sum_{i=0}^{|R(a)|} 1 \cdot r_i - \mathcal{A}(a)$$

Equation 9 – Per activity equality constraint where A is the set of all activities and a is an activity from this set, and $R(a)$ is the set of all resources which can be allocated to activity a and r_i is a resource in this set which is limited to the values one (1) for allocated and zero (0) for not allocated for all i. $\mathcal{A}(x)$ signifies the number of required allocations for activity x.

Just like with the constraint described in ***Equation 6***, this componentized constraint is only satisfied if and only if exactly the required number of resources ($\mathcal{A}(x)$) are allocated to the activity, which in this case is one (1). If no resources are allocated, then the result of the constraint equals -1 and if more than one (1) resource is allocated, then the constraint evaluates to ≥ 1. This constraint is expanded into allowing, or rather requiring, multiple resources to be allocated to an activity before it can be executed, by replacing the -1 component from ***Equation 8*** with $\mathcal{A}_x$ as seen in Equation 9.

By expanding the constraint of ***Equation 6***, into the smaller, per activity, constraints, the first constraint component also becomes linear.

The second constraint for resource allocation limitations, can also be redefined to translate into a linear problem. Just like with the one (1) (or n) resource(s) per activity, the constraint described in ***Equation 7*** has to be broken down into many smaller constrains on a per resource basis. Take for example the resource *Erik* from the input table in ***Table 6*** and its corresponding allocation budget from the resource allocation limit table in ***Table 5***. For this resource the constraint can be redefined as an equal-smaller than constraint (only satisfies when the constraint evaluates to zero (0) or smaller than zero (0)).

$$\iota_2 \in \{0,1\}, \iota_4 \in \{0,1\}: 0 \geq 1 \cdot \iota_2 + 1 \cdot \iota_4 - L$$

Equation 10 – Where ι_2 signifies whether Erik is allocated to activity "Payment is checked" (value of one (1)) or not (value of zero (0)), and ι_4 signifies whether Erik is allocated to activity "Request payment from customer" (value of one (1)) or not (value of zero (0)). The $-L$ component is obtained by obtaining the allocation limits from **Table 5***, in this case specifically for Erik.*

A more general description of this resource allocation $r \in R$ constraint for all resources can be described as follows:

$$0 \geq \sum_{i=0}^{|I(r)|} 1 \cdot \iota_i - L(r)$$

Equation 11 – Per resource allocation constraint where R is the set of all resources and r is a resource from this set, $I(r)$ is the set of allocation options of resource r where ι_i is an allocation option which is limited to the values one (1) for allocated and zero (0) for not allocated for all i. Finally, $L(x)$ signifies the allocation allowance for resource x.

This constraint differs slightly from the constraint described in ***Equation 7*** with the expanded condition that the result of the constraint may now also be lower than zero (0). Other than the expansion of the equality component, the constraint functions the same as the one in ***Equation 7***. The constraint is valid if and only if the number of times the resource is allocated is less than or equal to its allocation limit. If it is less, the constraint is satisfied (because the result is negative therefor smaller than zero (0)). However, if it allocated more than its allocation limit, then the constraint evaluates to a positive figure, thereby failing the less than or equal to zero (0) condition.

Finally, the last component is the **OBJECTIVE FUNCTION**. In **Objective Functions** three (3) objective functions are defined. Two (2) of these objective functions, being the *monetary cost* and *wait time* objectives, are potentially completely linear provided that of the two (2) cost-models discussed in **Cost Figure Variants** the linear regression model is used. However, the throughput time objective is not immediately linear, even if the linear regression model is used. When inspecting the objective function for throughput time in ***Equation 3***, two (2) components can be observed. The first component, $\Theta(r_0, r_1, \dots, r_{|A|-1}; A)$ is potentially strictly linear, again provided that the linear regression model is used, however the second component is not by virtue of it having a Max component.

Luckily, linearizing a Max component is not impossible and, in all actuality, not difficult to automize either as demonstrated in the following example [31], [32]:

Suppose the following expression with a max operator is to be linearized.

$$x = \text{Max}\{x_0, x_1\}$$

In this problem x has the following expected value $x \geq x_0, x_1$. This can be linearized in the following manner. A binary decision variable, γ, for which the value is dictated by the following relations, can be introduced:

$$\gamma = \begin{cases} 1, & x_1 < x_0 \\ 0, & x_1 \geq x_0 \end{cases}$$

Next a constant, C, is introduced for which holds that $x_0, x_1 \leq C$ within any legal and valid solution in the optimization problem. In the optimization problem in this thesis, C could be defined as the outcome of the worst-case scenario of allocations for a branch. This could be done by taking the sums of all resource-activity allocations in the different branches (where x_0, x_1 represent branches) such that no worse allocation can be chosen (in this case, the allocation limits introduces in **Determine legal resource-activity allocations** can be disregarded). Then the largest sum is chosen for C.

With the constant C and additional binary decision variable γ developed, an additional set of constraints can be defined to enforce that $x = \text{Max}\{x_0, x_1\}$ should it be linearized.

$$x \geq x_0$$

$$x \geq x_1$$

$$x \leq x_0 + C \cdot (1 - \gamma)$$

$$x \leq x_1 + C \cdot \gamma$$

With the above constraints, the max component is linearizable. However, this is an example with only two (2) elements in the max component. Looking at ***Equation 3*** more than two (2) elements per max component can occur. Should a max component contain more than two (2) elements, then this can be solved by substituting the elements with another max components and then linearizing the simplified elements as such:

$$x = \text{Max}\{x_0, x_1, x_2, x_3\}$$

$$x = \text{Max}\{\text{Max}\{\text{Max}\{x_0, x_1\}, x_2\}, x_3\}$$

However, this increases the complexity of the optimization problem, this also increases the additional binary decision variables needed, and it increases the number of constraints. The number of constants does not have to increase, because that simply must be *sufficiently* large enough. The number of required binary decision variables is $n - 1$ where n is the number of elements in a max component, and the number of constraints is $n + 2 \cdot (n - 1)$ (see **10.6** for a worked out example supporting this claim).

With the linearization of Max components, the objective function in ***Equation 3*** can also be linear, provided that it also only relies on linear regression for the cost, or rather throughput time, models. There also exist optimizers which themselves can deal with max components without the need to linearize first. Two (2) examples are in *Decision Optimization CPLEX* [33] and *GUROBI optimization* [34].

Taking the full deliberation on objective functions in mind, the optimization problem in this thesis can fall into two categories. The first category is *Integer Linear Programming*, and more specifically, *Binary-Integer Linear Programming*. If the objective functions are to be linear, then they must use the linear regression models for the cost models. If the objective functions are fully linear(ized) and the cost models are using the linear regression models, then this optimization problem becomes a NP-Complete problem by virtue of being linear and having the integer input set being limited to $\{0,1\}$ as described by *Karp's 21 NP-Complete problems* [30]. Otherwise, the optimization problem could be NP-Hard.

The second category is *Integer Non-Linear programming*, and more specifically for this thesis, *Binary-Integer Non-Linear Programming*. The optimization problem in this thesis will be non-linear in the first place if polynomial regression models are used for cost model construction.

3.1.8.5 Solver types

Knowing the (complexity) type of the optimization problem is not only useful for determining the complexity of the optimization problem, but also for determining which type of solver method should be used. As mentioned before, there are several existing optimization solvers, such as IBM CPLEX [33] and GUROBI [34]. The prototype for this thesis will solely use the IBM CPLEX solver. No research has been done into the exact difference and (dis)advantages between the two (2) optimizer frameworks, the IBM CPLEX optimizer was simply available at the time of writing.

When using an optimizer framework several *optimization engines/methods* are often available. This thesis will approach *mathematical programming* and *constraint programming* as optimization methods. Knowing the (complexity) type of the optimization problem, and knowing how the different

components, such as inputs, constraints, and objectives, are setup, is important for determining which optimization method/engine to use. The two (2) methods are compared below.

The first method which will be examined is the **MATHEMATICAL PROGRAMMING** method. Mathematical programming accepts both continuous variables, which can be limited to linear integer variables, thereby also to binary variables, as well as discrete variables. Mathematical programming requires a problem to lie within a well-defined mathematical problem type/category in terms convexity, linearity/non-linearity. Based on the mathematical problem type, the solver engine will choose a different solution strategy. Mathematical programming does not natively support all logical-arithmetic operators, such as minimum/maximum. In order to use minimum/maximum components, these components have to be linearized as has been described earlier on. The mathematical programming method is strictly limited to linear and/or quadratic problems [35].

Looking at the problem type described in **Problem Type**, an applicability conclusion can be drawn for mathematical programming. The mathematical programming engine can only use linear and/or quadratic problems. The objectives are only linear if all of its components are linear, which is only the case if linear models, as described in **Cost Figure Variants**, are used. So, in order to use mathematical programming, the optimizer must be limited to using linear cost models. Polynomial models could be quadratic, but only for small degrees. And, in order to retain the flexibility of the polynomial models, it is not recommended to combine the polynomial models with the mathematical programming engine.

The second method is the **CONSTRAINT PROGRAMMING** method. Constraint programming only accepts discrete input variables, such as Boolean or integer variables. Contrary to mathematical programming, constraint programming does not infer any assumptions on the mathematical problem type of the optimization problem, and it does not change its optimization strategy based on the different problem types. Constraint programming supports a broader range of logical-arithmetic operators as compared to mathematical programming, such as minimum/maximum, mathematical operators (addition, subtraction, division, multiplication, roots and exponents), modulo and operators such as logical comparators. Constraint programming is not limited to linear or quadratic problems, it can accept much greater exponents, but it is limited to discrete problems only [35].

When again looking at the problem type described in **Problem Type**, an applicability conclusion can be drawn for constraint programming. The main issue which could be a concern is the type of the input variables, luckily the input variables for the optimization problem in this thesis are all discrete in the form of binary variables. One (1) major difference between the constraint optimizer and the mathematical optimizer, is that the constraint optimizer can deal with greater than quadratic exponent expressions, so, both the linear as well as the polynomial models, as described in **Cost Figure Variants**, can be used without restrictions to the model degree depth.

The prototype for this thesis can use both the mathematical programming method, but only when the linear models are used, and the constraint optimizer for both linear and polynomial models. Which method is used for experimentation is described in **Runtime environment**.

3.1.9 Merge variant optimizations

In the last step, all variant optimizations have to be merged into one (1) final optimized resource and activity allocation combination set. Following the previous step from **Perform variant optimizations**, the optimizer should now have a set of activity-resource allocation tables. For each activity-resource allocation table, the variant for which the particular input table is optimized is recorded, along with its occurrence.

In merging the optimized activity-resource allocation tables, all activities for the entire process, not a pruned variant of this process, should have a resource allocated to itself. The merging of partial solutions from the variants can be done in several different ways. As mentioned before, the occurrence count of

variants can be taken into account in applying a certain importance weight to certain solutions. Different methods for merging and applying these importance weights are described in **Merge variant optimizations techniques**. This section will provide full a full overview of the proposed merging algorithms. For each optimization run of a process, one (1) of these methods is to be selected as the merging method of choice.

Once the results have been merged, the process optimizer will offer the user an allocation table where the set of available resource have been optimally allocated, within the defined constraints, to the processes' activities.

3.2 Cost Figure Variants

In **Find resource cost figures**, the concept of cost figure discovery is introduced. First the notion of cost needs to further elaborated. Cost in the context of cost figure, or cost model, discovery is in terms of the definitions set by the objectives and objective functions. Referring back to **Optimization definition** and **Objective Functions** three (3) objectives are defined, namely *monetary cost*, *throughput time*, and *wait time*. In the sense of cost figure discovery, cost refers to all three (3) objectives. Concretely, in cost figure discovery, *monetary cost* signifies the increase in monetary cost required to execute the process, when allocating a resource to an activity. Similarly, for throughput time, the cost is the increase in throughput time required to execute the process when allocating a resource to an activity and wait time cost is the increase in total wait time when allocating a particular resource to an activity in a process.

With cost defined, the method of assigning costs to allocations needs to be defined. In this thesis, regression models will be developed for the cost of allocation. Earlier in **Find resource cost figures** four (4) parameters for cost discovery and determination are defined, these parameters are key to developing the cost models. To simplify the model development, two (2) parameters, the activity, and the resource, can be approached as follows: instead of creating one (1), more complex, model for all activity and resource combinations, every viable resource and activity combination will receive a separate model. This eliminates the need for two (2) of the four (4) variables, only leaving the *frequency of resource allocation* and *activity occurrence count* in place. Note that, just like with regular regression exercises, the model can be expanded upon by introducing other variables to further strengthen these models. These variables could be anything which could impact a process, such as for example, weather, personnel satisfaction, budget, etc.

In this thesis, two (2) modelling types for cost figure discovery and modelling will be considered. One (1) type is linear regression, and the other type is polynomial regression. The next two (2) sections discuss and compares these two (2) options.

For both options, the regression is performed over the original event-data traces. The x variables which is used in the model construction are the number of times a resource is allocated, not just to the current activity, but to all activities, within a case and the number of times an activity occurs within a case. The target variable which it tries to predict is the observed objective-cost value. By observing all cases in a process-trace a set of independent resource-allocation and activity-occurrence values with a one-to-one mapping to the dependent cost-objective values is found. This resulting set can be used to construct regression models.

3.2.1 Linear regression

Linear regression is a method of regression which results in the construction of a linear model. The problem in this thesis cannot be solved with simple linear regression, as that only supports a single variable. Instead, multi-variate linear regression, or simply: linear regression, needs to be used. A linear regression model would be one such as the one represented in ***Equation 12***.

$$\beta_i \in B \forall i: f(X) = \sum_{i=0}^{|X|} \beta_i x_i + \alpha$$

Equation 12 – Linear Regression model where X is the set of all variables and x is a variable in this set, α is the y-intercept, and B is the set of all slope constants where β_i is a slope constant [36].

In the prototype for this thesis, and in testing, linear regression is implemented using the Python SciKit-Learn libraries [37].

Linear regression is an easy method of numerical model construction which does not have to require additional parameters. Unlike for example curve-fitting or polynomial regression, no target format, such as a target function [38], or poly-degree [39], has to be configured. For the implementation suggested in this thesis, all which is required is the base model as displayed in ***Equation 12***, and the three (3) regression input values which are the recorded cost figures, resource allocation counts and activity occurrence counts. Each model can also easily be explained by extracting the slope constants and y-intersect. Linear regression is also hypothetically a faster method as compared to other methods. Finally, as discussed in **Problem Type** use exclusive use of only linear models could simplify this optimization problem into an NP-complete instead of an NP-hard problem. However, the major downside of linear regression is that this model type only works if the data is mostly linear. If the data is not entirely linear, or if the data contains a non-linear section, then the linear regression method will probably fail to fit accurately.

3.2.2 Polynomial regression

The polynomial model is a linear summation of non-linear components (for *degrees* higher than one (1)). The modelling problem in this thesis, is a problem which by virtue of the unknown behaviour of input data curve lines makes for a strong polynomial regression candidate [39].

There are two (2) variants of the polynomial regression method which will be considered in this thesis. The first multi-variate model is represented in the formula seen in ***Equation 13*** and ***Equation 14***. This is a regular polynomial regression model, where there is no interaction between variables.

$$f_d(\vec{x}) = X\vec{\beta} + \alpha$$

Equation 13 – Polynomial regression, where d signifies the degree of the polynomial regression function, and α is the y-intercept [39]. *X and $\vec{\beta}$ are further worked out in* **Equation 14**.

$$f_d(\vec{x}) = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^{d-1} & x_1^d \\ 1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^{d-1} & x_2^d \\ 1 & x_3 & x_3^2 & x_3^3 & \cdots & x_3^{d-1} & x_3^d \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 & \cdots & x_n^{d-1} & x_n^d \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \end{bmatrix} + \alpha$$

Equation 14 – Polynomial regression where the $\vec{x}$ and $\vec{\beta}$ vectors are further worked out, where d signifies the degree of the polynomial regression function and n signifies then number of (independent) input variables.

The second model which will be considered for polynomial regression, is a model with variable interaction [39]. This model allows for the potential modelling of relations between different variables. Interaction regression makes for more complex expressions, and to limit the complexity of the expressions, only a definition for a two-variable model will be provided. This two-variable polynomial regression model with interaction is shown in ***Equation 15***.

$$\exists \beta_0 \in B: f(x_0, x_1, d) = \beta_0 + \sum_{i=0}^{d+1} \sum_{(d,b_d) \in E(B_{[\mathcal{D}(i-0):\mathcal{D}(i)]})} \begin{cases} \mathscr{b} x_0^{i-d} x_1^d, & i-d > 0 \wedge d > 0 \\ \mathscr{b} x_0^{i-d}, & i-d > 0 \wedge d \leq 0 \\ \mathscr{b} x_1^d, & i-d \leq 0 \wedge d > 0 \\ 0, & i-d \leq 0 \wedge d \leq 0 \end{cases} + \alpha$$

Equation 15 – Polynomial regression where x_0, x_1 are input variables and B is the set of all slope constants where β and $\mathscr{b}$ are slope constants, and α is the y-intercept. $E(s)$ is an enumerator function which returns two (2) sets, the first is a set of counters from 0 to n where n is the size of the second set, and the second set are the objects from the original input set s. $B_{[a:b]}$ signifies a subset of B from index position a to index position b. $\mathcal{D}(x)$ calculates the number of coefficients for a given degree x for a polynomial regression model with interaction over two (2) variables. The full definitions of $\mathcal{D}(x)$ is given in **Equation 16**.

$$\mathcal{D}(n) = \frac{(n+2)(n+1)}{2}$$

Equation 16 – Calculates the number of coefficients for a given degree n for a polynomial regression model with interaction over two (2) variables.

If it is known beforehand that the two (2) variables are completely independent, then the non-interactive variant should be chosen to reduce model complexity and increase explainability. The size of the expression with regards to the number of variables and the degree depth increases linearly, whereas the expression with interaction on two (2) variables grows quadratically (see ***Equation 16***). However, in the context of this thesis, it is not a guarantee that variables are completely independent. Processes, and the relation the different variables have with one and another, can be wildly different between use-cases. For this sole reason, the unpredictability of the data and the unpredictability of relations in the data, the more flexible method is chosen, sacrificing readability of the created models.

In the prototype for this thesis, and in testing, polynomial regression is implemented using the Python SciKit-Learn libraries [37]. Besides using the library for the base implementation, a few extra steps elements are used to for example tune the degree hyper parameter, and to prevent overfitting.

3.2.2.1 Degree fitting

Finding the best degree should be done by first splitting the data in a training and testing set [40], and then use these to model for different degrees on the training data and observe the *Coefficient of Determination* and *Root Mean Square Error* score on the testing data.

The *Coefficient of Determination* (R^2 see ***Equation 17***) [41] and the *Root Mean Square Error* ($RMSE$ see ***Equation 18***) [42], where the closer the R^2 between the target data and the model data to the value of one (1) gets, the more accurate the model is in terms of how well the predicter variables can explain the variance in response variables. And the smaller (closer to zero (0)) the $RMSE$ between the target and model data gets, the closer the model output lies to the target data.

$$R^2 = 1 - \frac{\sum_{i=0}^{n} \big(y(i) - f(i)\big)^2}{\sum_{i=0}^{n} (y(i) - \bar{y})^2}$$

Equation 17 – R^2 where the quotient between the squared sum over the error between the target and regression line, and the squared sum over the error of the mean line, are subtracted from one (1). y is the observed data, $\bar{y}$ is the mean line over the observed data, and f is the result from the regression function [41].

$$RMSE = \sqrt{\frac{\sum_{i=0}^{n} \big(f(i) - y(i)\big)^2}{n}}$$

Equation 18 – RMSE where the sum of the squared error between the regression value and the target value is divided by the number of samples to obtain the mean, which is squared to compensate the squared component. y is the observed data, and f is the result from the regression function [42].

When the data becomes overfitted, and thereby the degree being too large, the R^2 should start to drop off from its highest observed value, and the $RMSE$ should start to increase again from the lowest observed value.

3.2.2.2 Multiple Model Issue

In this particular optimization problem, as described in this thesis, where for each activity-resource combination a model has to be generated, one (1) aspect to keep in mind is that not every model for an activity-resource combination, will have the same optimal *degree*. There are several options to deal with the multiple model issue. Each of these options is a compromise between accuracy and speed. The most optimal degree will be defined as is stated before, the degree which produces the highest R^2 score and the lowest $RMSE$ score on test data.

- **SINGLE DEGREE** This option sets a one (1) degree for all models. This hypothetically should allow for the fastest model construction, but it could be at the cost of accuracy on a per model basis;
- **BEST AVERAGE DEGREE** This option finds the best average degree out of a subset of all activity-resource combinations. For a subset, say for example a random set of 20% of the activity-resource combinations, determine their most optimal degree. For these activity-resource combinations, the particular model and degree can be saved regardless of the next step. Once for each of the combinations the most optimal degree is found, the best average is found by averaging all R^2 scores and $RMSE$ scores and picking the degree with the best average score. Hypothetically this should provide a better overall fitting model than the *single degree* option, but it is slower to run because multiple models have to be developed;
- **BEST n-AVERAGE DEGREE** An extended version of previous option would be to use a best n-average degree. In this option, instead of one (1) degree being used for all other models, n-degrees could be chosen from the average pool. When modelling the other activity-resource combinations, the best fitting degree could be chosen out of the n-degrees from the average pool. Hypothetically this allows for a greater degree of freedom and possibly better fitting degrees, but this increases complexity because for each model, n different degrees need to be tried;
- **PER MODEL BEST DEGREE** The final option is to determine the best degree on a per model basis. For each model, find the best degree, out of the available range, and use that for that particular model. Hypothetically, this should allow for finding the best fitting degree across all models, however this is also computationally the most expensive, because each activity-resource combination needs to be modelled multiple times and evaluated on their best degree. There are however two (2) methods of going about degree search in this option:
    - **EXHAUSTIVE BEST DEGREE SEARCH** Develops every degree option into a model, and then finds the best configuration. This is the most expensive option;
    - **GREEDY BEST DEGREE SEARCH** Keeps developing models for all the available degree options, until a (local) convergence occurs based on the highest-yet R^2 score and the lowest-yet $RMSE$ score. Once a degree is found where the R^2 is decreasing again and the $RMSE$ score is increasing, it will stop its search, with the assumption that after an overfit, the increase in degrees will never reduce the overfitting. This could potentially save on computation time.

3.2.3 Exhaustive fit on best model

When developing hyperparameters, such as the degree setting in the polynomial regression method shown in **Multiple Model Issue**, the hyperparameters are not developed on the entire dataset. Instead, the data will be split up in a train dataset and a test dataset. This is done to prevent overfitting to the target data and skewing test results. A model should be a generalistic model for the target data, so even if it is trained on a subset of the target data, it should still score well on a different subset.

However, once the hyperparameters have been set up correctly, a model can be trained on the entire dataset. This is done to feed it more, and more variate data, which should make it more robust and generalizable. Both implementations of the modelling methods described in this thesis have the ability to train on the entire dataset, instead of a smaller split. For linear regression this simply means that it will immediately fit on the entire dataset. For the polynomial regression model, depending on the degree determination mode, it will (re)train its model after the final degree has been found.

3.3 Variants to optimize selections

In **Perform variant optimizations** the notion of selecting variants for optimization was introduced. In this section, three (3) different methods for selecting variants to optimize will be elaborated on. Each of these methods will most likely be a trade-off between compute time performance and accuracy.

3.3.1 All selection

The first option would be to simply optimize for every variant and pool all results into the merging stage. This might however, depending on the merging strategy (see **Merge variant optimizations techniques**), lead to *useless computation*, as for some merging strategies, the least prevalent variants impact hardly impact the overall result. On the other hand, this method could hypothetically provide the most realistic outcome. It might not provide the highest performing outcome, but since it also takes niche cases into regard, it could hypothetically prevent edge case/niche case performance bottlenecks.

3.3.2 Minimum requirement selection

A second option is to only select as many requirements as are required. The selection should be done based on prevalence. The variant selection step would continuously select an extra variant into the variant-optimization-pool, starting at the most prevalent variant and moving towards the least prevalent, until all activities which exist in the complete process also exist in the variant-optimization-pool.

The pseudo algorithm for this algorithm can be defined as follows:

```
target_set := {∗ A_p, };
selected_variants := [∅, ];

for (v in Vs_p) {
  selected_variants.append(v);

  if (|target_set ≔ target_set − {∗ A_v}| < 1) {
    break;
  }
}

return selected_variants;
```

Figure 7 – Pseudo algorithm describing the minimum requirement selection method. A_p is a list of all activities occurring in process p, and the $*$ operator in front of a list unrolls said list into individual components. Vs_p is the set of all variants of process p which are sorted from highest to lowest occurrence, and v is a variant in this set. A_v is a list of all activities in a given variant v.

This could have several benefits. First, the computation time is reduced, because only a subset of all variants is optimized. Additionally, because only the most prevalent variants are optimized for, the resulting model will be focussed on the highest performance for the majority of cases.

3.3.3 Minimum requirement selection non repeating

Finally, a third option can be proposed, which is an extension on the *minimum requirement variant selection method* above. The minimum requirement selection non repeating method is similar in

behaviour in that it only searches and appends variants to the selected variants target list until all activities in a process have been found. But where it differs is in the *non repeating* part: it avoids adding more variants than necessary by only adding variants to the pool if a variant contains an activity which has not been covered before. If, in its search, it comes across a variant which does not contain activities which have not been seen before, it will not be added to the pool.

The pseudo algorithm for this algorithm can be defined as follows:

```
target_set := {∗ A_p, };
selected_variants := [∅, ];

for (v in Vs_p) {
  if (|target_set ∩ {∗ A_v}| > 0) {
    selected_variants.append(v);

    if (|target_set ≔ target_set − {∗ A_v}| < 1) {
      break;
    }
  }
}

return selected_variants;
```

Figure 8 – Pseudo algorithm describing the minimum requirement selection nonrepeating method. A_p is a list of all activities occurring in process p, and the $*$ operator in front of a list unrolls said list into individual components. Vs_p is the set of all variants of process p which are sorted from highest to lowest occurrence, and v is a variant in this set. A_v is a list of all activities in a given variant v.

The primary advantage of this approach is to minimize the size of the pool, thereby minimizing the number of optimizations that have to be run. A downside to this approach is that it could occur that a lesser occurring variant is "overrepresented" in the pool simply by having a rare activity. Other variants which might occur (far) more often might be skipped simply because the activities they cover have already been covered by other, previous, variants.

3.4 Merge variant optimizations techniques

Optimization of processes in this thesis is approached with a divide and conquer approach by dividing the problem into several smaller optimization problems. As stated in **Perform variant optimizations**, a (sub)set of variants are each individually optimized. After optimizing, each variant's solution needs to be merged into a general solution, as stated in **Merge variant optimizations**. In this section, several techniques for ordering the *importance* of variants are discussed. Experiments on these different techniques are set up in **Variant selection and merge methods comparison**.

However, first the idea of merging needs to be elaborated on. For each of the optimized variants, there exists a resource-activity allocation table indicating the most optimal allocation for that variant. These tables need to be merged into a single complete resource-activity allocation table for the entire process. For each activity, there is a set of resources, with a size of $n \geq 1$, obtained from the different variants, from which the best resource candidate needs to be derived.

Determining the best resource for each activity, is where the following three (3) different techniques are used. All techniques rely on a *largest vote percentage* basis, where the resource with the largest allocation-percentage is allocated. For example, in the following set: $V = \{a: 30\%, b: 20\%, c: 15\%, d: 15\%, e: 10\%, f: 8\%, g: 2\%\}$ a would be chosen by virtue of having the largest percentage of *votes*. For all three (3) techniques, this principle is identical, the different techniques only determine how the *votes* are allocated. For all examples in the three (3) techniques, ***Table 7*** is used as the *input*.

Table 7 – Example scenario portraying six (6) different variants all proposing a resource for four (4) activities. The occurrence count indicates how often a variant occurs in the total process event-dataset. V_i indicates a variant, r_i indicates a resource proposed by the variant (in the same column) for an activity, and a_i indicates an activity for which resources are proposed.

	V_0	V_1	V_2	V_3	V_4	V_5
Occurrence count	100	75	50	30	20	10
Occurrence percentage	35%	26%	18%	11%	7%	3%
Activity	**Resource**					
a_0	r_1	r_2	r_0	r_3	r_0	r_0
a_1	r_4	r_5	r_6	r_5	r_7	r_8
a_2	r_9	r_{10}	r_{10}	r_{11}	r_{12}	r_{13}

3.4.1 Highest count merging

The first technique, highest count merging, looks at how often a resource is put forth by the variants. In this technique, the variant occurrence is not taken into account. This technique should hypothetically prevent less prevalent variants from being disregarded outright. With preventing smaller variants from being disregarded, a more favourable balance towards preventing worst case process scenarios could hypothetically be created.

For any given activity, the average merging technique will count for each proposed resource how often it is proposed by the variants for that particular activity. The resource which is proposed the most will be the resource which will be allocated to the activity. If a tie occurs between two (2) or more resources, then the resource which occurs first, sorted by variant occurrence, will be proposed for allocation. Looking at the example in ***Table 7***, if for activity a_0 resources r_1 and r_2 would tie on count, then resource r_1 will be proposed.

When applying the highest count merging technique on ***Table 7***, then the following result is to be expected:

Table 8 – Result table of the highest count merging technique applied on the input from **Table 7***. Under Resources and counts, the notation $r_i{:}\,C$ indicates with r_i the resource and C the respective count for that resource.*

Activity	Resources and counts					Allocated resource
a_0	$r_0{:}\,3$	$r_1{:}\,1$	$r_2{:}\,1$	$r_3{:}\,1$		r_0
a_1	$r_4{:}\,1$	$r_5{:}\,2$	$r_6{:}\,1$	$r_7{:}\,1$	$r_8{:}\,1$	r_5
a_2	$r_9{:}\,1$	$r_{10}{:}\,2$	$r_{11}{:}\,1$	$r_{12}{:}\,1$	$r_{13}{:}\,1$	r_{10}

3.4.2 Weighted average merging

In the second option, weighted average merging, variant occurrence is considered for determining which resource should be allocated. This technique hypothetically helps to allocate more effectively for a better average case because variants which are more prevalent weigh in more for the allocation decision. This better average performance is to be expected because historically based on the event-data there is a higher chance that the more prevalent variants occur again, and with the final allocation being more suited for these more prevalent variants, the performance for these variants should increase.

The weighted average merging works by taking the weighted average, based on occurrence count, for each resource, and selecting the resource with the highest weighted average. The weighted average for a resource is calculated as follows:

$$\tilde{r}_{a,i} = \frac{\sum_{j=0}^{|P_{a,r}|} p_j}{N}$$

Equation 19 – Where $P_{a,r}$ is the set of occurrence counts for the variants for a given resource r and activity a and p is an occurrence count within this set, $\tilde{r}_{a,i}$ is the weighted average for resource i for activity a, and N is number of total cases (the sum of all occurrence counts for all variants).

When considering the example data in ***Table 7***, and focussing on activity a_0 and resources r_0 the weighted average score for r_0 is:

$$P_{0,0} = \{50, 20, 10\}$$

$$N = 285$$

$$\tilde{r}_{0,0} = \frac{80}{285} = 0.281$$

A tie occurrence in the weighted average, will be dealt with in the same manner as in the *highest count merging* method. Expanding this principle to the entire example data, the following results are to be expected:

Table 9 – Result table of the weighted average merging technique applied on the input from **Table 7**. *Under Resources and counts, the notation $r_i : C$ indicates with r_i the resource and C the respective weighted average.*

Activity	Resources and weighted average					Allocated resource
$\boldsymbol{a_0}$	r_0: 0.281	r_1: 0.351	r_2:0.263	r_3:0.105		$\boldsymbol{r_1}$
$\boldsymbol{a_1}$	r_4: 0.351	r_5: 0.368	r_6: 0.175	r_7: 0.070	r_8: 0.035	$\boldsymbol{r_5}$
$\boldsymbol{a_2}$	r_9: 0.351	r_{10}: 0.439	r_{11}: 0.105	r_{12}: 0.070	r_{13}: 0.035	$\boldsymbol{r_{10}}$

3.4.3 "Pareto" merging

The third option is an option highly based around the Pareto principle of *"80% of the results stems from 20% of the effort"* [28]. In the context of merging techniques, this principle will be applied to heavily skew the allocation favourably to the more prevalent variants. This amplifies the hypothesized effect discussed in the weighted average merging technique.

Instead of using the occurrence of a variant as the weight, the occurrence now dictates the position of the variance on the distribution line of the *Pareto Distribution* with the *Probability Density Function* (see ***Equation 20***)[43]. The distribution of the variance is achieved by normalizing the occurrence over the input with *min-max feature scaling* (see ***Equation 21***). However, this scaling will be reversed, such that the highest occurrence will obtain the lowest normalized value.

$$pdf_{\alpha,\beta}(x) = \begin{cases} \alpha \cdot \dfrac{\beta^{\alpha}}{x^{\alpha+1}}, & x \geq \beta \\ 0, & x < 0 \end{cases}$$

Equation 20 – Pareto Probability Density Function, where pdf is the result of the function, x is an input on the distribution line, α being the shape parameter and β being the scale parameter [43].

$$x_{n_{X_+,X_-,S_+,S_-}}(x) = \frac{X_+ - x}{X_+ - X_-} \cdot (S_+ - S_-) + S_-$$

Equation 21 – Reversed min-max feature scaling where x_n is the normalized value of the input x, X_+ is the highest value of all the input values, X_- is the lowest value of all input values, S_+ is the highest target scale value, and S_- is the lowest target scale value.

Between the *Probability Density Function* and the *reversed scale min-max feature scaling*, the two scale factors, β, and S_+ and S_- respectively, are to be linked. In this implementation, a scale of $\beta = S_+ = 10$ will be chosen for the upper bound of the *Probability Density Function* input, and the lower bound will be set to $S_- = 1$. This means that all occurrence counts from ***Table 7*** will translate to normalized input values in the domain D: $[1,10]$. Applying the normalization, the following results for occurrence counts should be expected:

Table 10 – Normalized occurrence counts with the values from **Table 7** *as inputs.*

	$\boldsymbol{V_0}$	$\boldsymbol{V_1}$	$\boldsymbol{V_2}$	$\boldsymbol{V_3}$	$\boldsymbol{V_4}$	$\boldsymbol{V_5}$
Occurrence count	100	75	50	30	20	10

Normalized occurrence count	1	3.5	6	8	9	10

Another value to set in ***Equation 20*** is the shape α. For this thesis a simplified shape value of $\alpha = 1$ will be used, however this hyper parameter, along with the scale parameter β could be tuned. The resulting *Probability Density Function* (see ***Equation 22***) will be applied over the inputs from ***Table 7***.

$$pdf_{1,10}\left(x_{n_{100,10,10,1}}(x)\right) = \frac{10}{x_{n_{100,10,10,1}}(x)^2}$$

Equation 22 – Pareto Probability Density Function with a reverse min-max scaling applied where $X_+ = 100$, $X_- = 10$, $S_+ = 10$ and $S_- = 1$, and $\alpha = 1$ and $\beta = 10$. Note that the $x < 0$ (or rather $x_n < 0$ in this case) from **Equation 20** *is omitted because the scaling provided by x_n has a lowest bound of $x_n = 1$.*

The application of the *Probability Density Function* and its configuration are done by calculating for each resource the sum of all *pdf* values based on the occurrence counts of the variants putting forth the resources for a particular activity. The resource with the highest accumulated *pdf* values will be allocated for an activity. Tie occurrences are dealt with in the same manner as in the *highest count merging* method and *weighted average merging* method. The accumulated *pdf* values for a resource are calculated as follows:

$$\hat{r}_{a,i_{\alpha,\beta,X_+,X_-,S_+,S_-}} = \sum_{j=0}^{|P_{a,r}|} pdf_{\alpha,\beta}\left(x_{n_{X_+,X_-,S_+,S_-}}(p_j)\right)$$

Equation 23 – Where $P_{a,r}$ is the set of occurrence counts for the variants for a given resource r and activity a and p is an occurrence count within this set, $\hat{r}_{a,i}$ is the sum of the pdf values for resource i for activity a, $pdf_{\alpha,\beta}$ is the Pareto Probability Density Function from **Equation 20** *and $x_{n_{X_+,X_-,S_+,S_-}}$ is the Reversed min-max feature scaling from* **Equation 21**.

When considering the example data in ***Table 7***, and focussing on activity a_0 and resources r_0 the accumulated *pdf* value for r_0 is:

$$P_{0,0} = \{50, 20, 10\}$$

$$\hat{r}_{0,0_{1,10,100,10,10,1}} = 0.278 + 0.123 + 0.100 = 0.501$$

Applying ***Equation 23***, with the configurations shown in ***Equation 22*** (where the normalized values are presented in ***Table 10***), to the entire example data, the following results are to be expected:

Table 11 – Result table of the weighted average merging technique applied on the input from **Table 7**. *Under Resources and counts, the notation $r_i : C$ indicates with r_i the resource and C the respective sum of the pdf values.*

Activity	Resources and pdf sum					Allocated resource
a_0	r_0: 0.501	r_1: 10.0	r_2:0.816	r_3:0.156		r_1
a_1	r_4: 10.0	r_5: 0.972	r_6: 0.278	r_7: 0.123	r_8: 0.100	r_4
a_2	r_9: 10.0	r_{10}: 1.09	r_{11}: 0.156	r_{12}: 0.123	r_{13}: 0.100	r_9

3.5 Multi-objective optimization

The process optimization method described in this thesis is designed to exist within a business environment. More specifically, the optimization method should optimize business processes and supply chain processes. In the introduction, **Optimization definition**, several objectives have been defined. Up and until now, the proposed method has been primarily focussed on single objective optimization. However, within a business environment, single objective optimization or rather prioritisation, is hardly favourable. Ideally all objectives are optimized to their theoretical best but depending on circumstances this is not always possible. Therefore, multi-objective optimization exists as a compromise between objectives. More specifically, it exists as a compromise between objectives on a *Pareto optimum front*, where improving one objective probably results in worsening the other objective. Given this setup, this optimizer should apply a technique which allows for multi-objective optimization within the environment, goals, and constraints setup within this thesis.

3.5.1 Multi-objective optimization applicability

Before looking at the specific possible approaches, and specific possible implementation options, a case should be reviewed to show that multiple objectives could have compromising effects on one and another. In this thesis a method for constructing test-, or rather experiment-process-traces will be proposed. This method of generating experiment-process-traces will also be used in this section to demonstrate that within certain environments, a possible compromise, with an accompanying Pareto optimum front, between objectives can occur.

In **10.7** a fully detailed setup of the trace generator parameters is defined. Here a short description of the generated model will be provided. In order to demonstrate a solution space where two objectives do not necessarily provide an absolute optimum, where both objectives are optimally minimized, a simple process is set up with only four (4) activities. Two (2) of these activities are sequenced activities, and two (2) activities are looped activities. A visual representation of the process can be seen in ***Figure 9***.

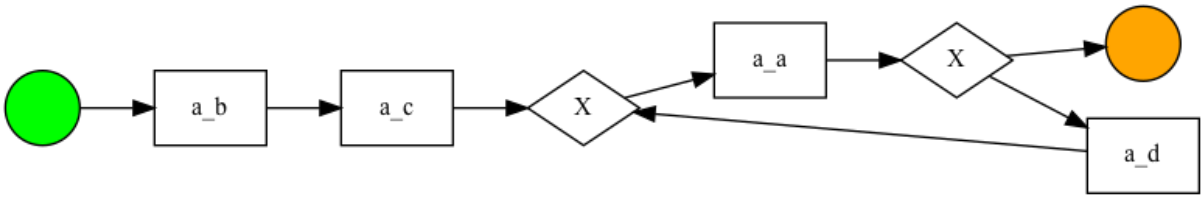


Figure 9 – Pareto Example Process

Besides the process, a set of resources is required to start generating a full trace. In order to create a problem space which offers a compromise between multiple objectives, several sets of resources are to be generated. These different sets of resources are each "specialized" in a specific objective. Note that "specialized" in this context means that the simulation base values and certain modifiers are favoured for a particular objective. In this specific case, three sets of resources are generated. For this example, one set is specifically generated to have more favourable properties for *throughput time*, one is specifically generated to favour *cost performance*, and one set is a balanced performer for both objectives, but in both cases worse than the "specialized" resources for one particular objective. These resources are similar in setup as the ones which will be used in the experimentation. Please refer to **Resource simulation values**, to find a full justification for the reasoning and "real-world-comparing" for these resources.

The combination of the process and set of resources results in a process trace containing a variety of different process variants, and activity-resource combinations. Specifically, 1000 different cases, or instances, of the above process has been simulated, which resulted in five (5) different variants, and 23 unique activity-resource combinations.

To show the applicability of multi objective optimization on this particular case, it should be demonstrated that the solution space shows a compromise between the two (2) objectives. This means that, in order to show this, the (entire) solution space should be developed. Developing the solution space first requires developing the models which are going to provide the throughput and cost results based on their application to the process. The models which are developed for this applicability showcase, are based on polynomial models, as described in **Polynomial regression**, where the per model degree is developed according to the exhaustive per model best degree approach as introduced in **Multiple Model Issue**. After these models have been developed, they can be used to develop the complete solution space. Note that for this example, only the complete solution space for a found variant of this process will be developed. The variant which is developed is the second variant, which has a pruned process tree of the shape as shown in ***Figure 10***. This variant displays two (2) instances of the loop *(activities 'a_a' and 'a_d')* as seen in ***Figure 9***.

```
('>',
 ['a_b', 'a_c', 'a_a',
  'a_d', 'a_a', 'a_d',
  'a_a'])
```

Figure 10 – Pruned second variant process tree of the process in Figure 9

Developing this solution space is accomplished by simulating every possible, legal (as defined by its constraints set in **Constraints**), combination of resources and activities through the objective functions from **Objective Functions** and then recording the results from the objective functions. This results in the following solution space:

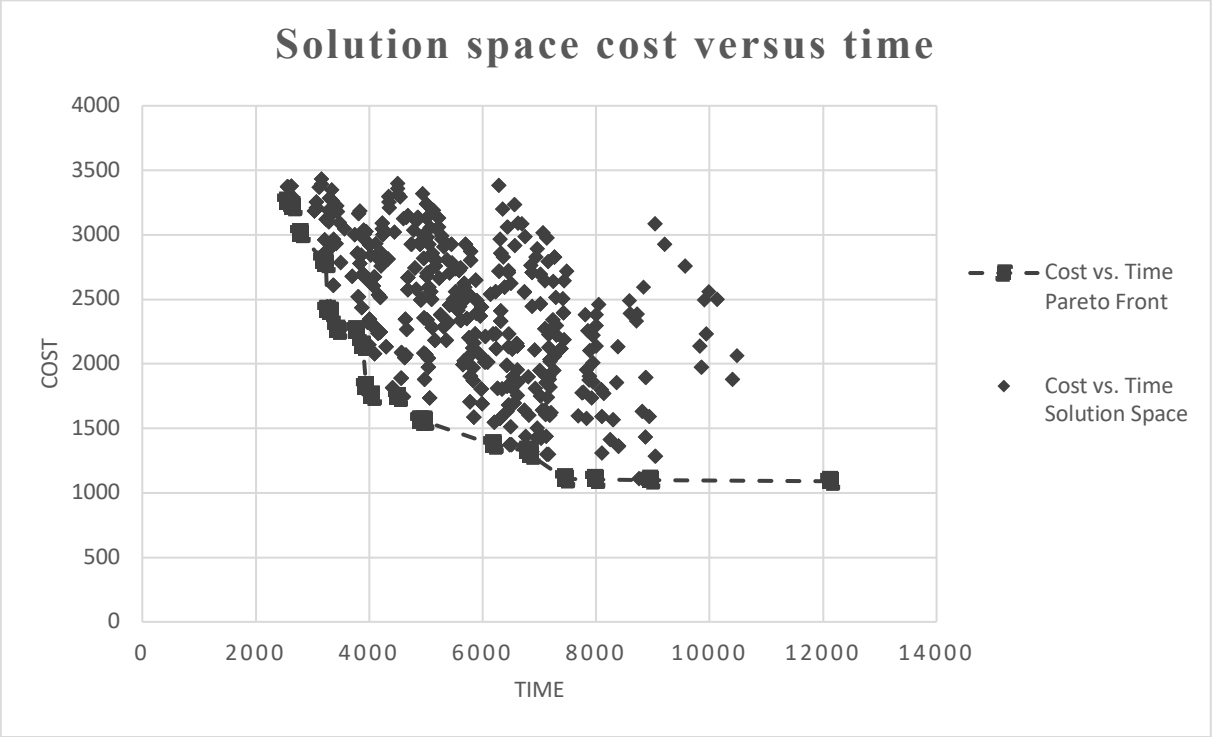


*Figure 11 – Pareto Optimum front demonstration for the process in **Figure 9**. Please refer to **10.8** for a larger version.*

This fully developed solution space shows every feasibly and legal possible solution outcome for the second variant of the above-described example process. The dashed line in ***Figure 11*** illustrates the Pareto optimum front where all dominating (or rather non-dominated) solutions exist. Sliding across this line would give an optimum compromise between the two (2) objectives.

The solution space in this applicability example shows that for this particular case a multi-objective optimization strategy is required to get "acceptable" results. One point of discussion is how well this translates to real world applications. This example has been specifically generated to create a compromised solution space; however, it is not entirely unrealistic to assume that a real business and/or supply chain application could have specialized resources. Within a real-world application, a set of resources could exist which are highly specialized for high throughput realizations (meaning low throughput time), but which are expensive because of it, and on the other hand, a set of resources could exist which are specialized in cost excellence, but therefor lack in raw throughput capabilities. So, despite this being a specifically simulated and crafted scenario, the possibility of the existence of specialized resources does merit the need for multi-objective optimization strategies.

3.5.2 Multi-objective optimization techniques

In the above subsection, the need for multi-objective optimization techniques and strategies has been laid out. In this subsection, different techniques will be evaluated for the optimization method described in this thesis. In the next subsection, the implementation of the best fitting technique will be laid out.

The main goal of multi-objective optimization is to find a solution somewhere in the solution space, which is a dominating solution, or in other words, which lies on the Pareto optimum front. This means that for a given dominating solution, no other combination can be found where both objectives improve. The only manner of improving one objective in this sub-set of solutions is by compromising on the other objective. Ideally a multi-objective technique is able to slide along the Pareto optimum front by adjusting some parameter determining the importance weighing of the objectives.

In order to evaluate different techniques, an acceptable set of minimum requirements needs to be defined. This set of requirements needs to work within the confines of the problem statement and proposed method of this thesis. As described throughout previous sections, this thesis aims to propose a generalizable method for optimizing a (business) process and/or supply chains by means of event-log analysis. This brings out a problem specific to this thesis: the problem, and therefor solution space, may be completely different between different cases thrown at the optimizer. The exact shape and behaviour of the objective models is not known before the start of the optimization process. More precisely, the shape and behaviour of the models are only known after all the steps of the optimization process method have been executed. Despite this, the optimizer should work regardless of what shape or type of behaviour the models, derived from the input trace, show. It should optimize regardless of knowing the solutions space it can work in. With the possibility of the models being completely different between cases, the solution space can be completely different in shape and properties between cases as well. A solution space could have a single point best objective with only one (1) dominating solution, or a Pareto optimum front with multiple dominating solutions, furthermore, the solution space could be convex, and it could also be non-convex. Lastly, the absolute min and max values of an objective are not known beforehand. This brings out the following requirements within which the multi-objective optimizers need to work in:

- **NO (ABSOLUTE) MIN MAX KNOWLEDGE** The multi-objective optimization techniques need to be able to work without having any prior knowledge of the absolute min and max values of each objective;
- **CONVEX AND NON-CONVEX** The multi-objective optimization techniques must be able to handle both convex and non-convex solution spaces;
- **NO KNOWLEDGE PARETO FRONT SLIDING** Without any (prior)knowledge of the shape, domain, range and behaviour of the underlying models, the multi-objective optimizer should be able to offer a *sliding* parameter which allows for sliding over all, or most, possible dominating solutions;
- **SCALE INDEPENDENT** Finally the multi-objective optimization technique needs to be able

In this thesis, only classical multi-objective techniques will be evaluated, as at least one of these techniques (as described further on) suffices in achieving an acceptable multi-objective optimization strategy for this particular use case. Whilst genetic multi-objective optimization algorithms offer certain advantages such as providing a set of solutions instead of a single point solution [44], in this thesis the scope multi-objective optimization is limited to exploring whether multi-objective optimization is possible within this particular problem statement and proposed method. However, in future research, genetic algorithms can also be explored within the same requirements/constraints as described above. The primary goal and added value of genetic algorithms would then be to develop a set of optimal solutions instead of a point.

The classical multi-objective optimization techniques which will be explored are **THE WEIGHTED SUM METHOD**, **THE ε-CONSTRAINT METHOD**, and **THE WEIGHTED METRIC METHOD**.

3.5.2.1 Weighted Sum

The weighted sum method is a method where multiple objective functions are combined into one (1) single objective function by means of applying weights. The collective objective function is defined as follows:

$$w \in W\text{:minimize}\big(F_O(x)\big) = \text{minimize}\left(\sum_{o=0}^{|O|} w_o f_o(x)\right)$$

Equation 24 – Weighted sum of all objective functions where O is the set of all objectives and o is an objective from this set, where W is the set of respective weights to be applied to each objective from O and w is a weight from this set, x are the input variables or single input variable for all objective functions from O.

The primary advantage of the weighted sum method is its simplicity of implementation. No prior knowledge about the solution space and objectives is required to get this method to work. The only aspect which needs to be set is the weights to apply to each objective in order to make this multi-objective optimizer work. However, with its simplicity comes several disadvantages [44], [45].

The first disadvantage is the difficulty of setting the weights. When the objective functions have entirely different scales, the weights must be adjusted for that. Another element which needs to be considered is the magnitude of outcomes produced by the objective functions. If one objective function for example has an outcome difference of lets say ten (10) between solutions, and another objective has an outcome difference of lets say 1000 for the same solutions, then the weights need to be adjusted for this. If weights are set incorrectly, the found solution after optimization might not even be a dominating solution on the Pareto optimum front.

Even though earlier it was stated that *"No prior knowledge about the solution space and objectives is required"* to get this method to work, this is only true for just getting it to work. For getting this method to work effectively and acceptably, so to produce solutions on the Pareto optimum front, the exact behaviour and shape of models and objective models is required, which goes against one of the requirements.

Another problem with the weighted sum method, is that it cannot find some solutions on the Pareto optimum front in case of a non-convex solution space. This is also problematic, as some process cases might produce non-convex solution spaces, as is for example the case in ***Figure 11***.

3.5.2.2 ε-Constraint

The ε-constraint method, unlike the other two (2) methods does not alter the expression which will be optimized for by minimization. The ε-constraint achieves multi-objective optimization by means of introducing an additional constraint. This additional constraint is essentially an expression representing one of the objective functions, which will be limited to a certain maximum value, which is the ε parameter.

As the optimization functions are not changed, the actual optimization expression does not change as compared to single-objective optimization. However, for the constraints, the following expression can be defined:

$$\forall o \in O^{-}, \forall \epsilon \in \mathrm{E}\text{:}\, f_o(x) \leq \epsilon_o$$

Equation 25 – ε-constraints for all but one(1) objective function, where O^{-} is the set of all objective functions, with the exception of the primary target objective function and o is an objective function from this set, where E is the set of all constraining epsilon parameters respective to the set of objectives and ϵ is an epsilon constraint from this set, x are the input variables or single input variable for all objective functions from O^{-} and also including the primary target objective.

The primary advantage of the ε-constraint method is that it works with both convex and non-convex problems, and it can be scale and magnitude independent if epsilon is chosen cleverly [44], [46]. In this method, it is important to carefully choose the epsilon value for every objective function, as this determines the range of (dominating) solutions this objective function will accept. This does require some knowledge of the feasible solution range, however this can also be approximated. And, unlike the weighted sum method, a poorly chosen epsilon value does not result in a dominated solution, so, as long as the epsilon values do not constrain outside of the solution space, an optimal solution on the Pareto optimum front should always be found.

3.5.2.3 Weighted Metric

The weighted metric method, or the Tchebycheff method, again tries to combine multiple objectives into one (1) single objective to optimize for. The weighted metric method seems similar to the weighted sum method, in that weights are applied to every objective function, however, before weights are applied, the objective functions are adjusted with the difference between them and the vector elements of an ideal solution. Lastly there is a scaling factor which determines how the final objective function is shaped. The objective function is constructed as follows:

$$w \in W, z^* \in Z^\star, D_p\colon [1,\infty)\text{:minimize}\big(F_O(x)\big) = \text{minimize}\left(\sum_{o=1}^{|O|} w_o |f_o(x) - z_o^\star|^p\right)^{\frac{1}{p}}$$

Equation 26 – Weighted metric sum of all objective functions where O is the set of all objectives and o is an objective from this set, where W is the set of respective weights to be applied to each objective from O and w is a weight from this set, x are the input variables or single input variable for all objective functions from O, and $Z^\star$ is the ideal solution vector of which $z^\star$ are elements in this ideal solution vector, and finally p is a scaling vector.

Of the three (3) methods, the Tchebycheff method is the most "complete" method, as it guarantees that all Pareto optimum solutions can be found with a given ideal vector $Z^\star$ [44], [47], [48]. It is also able to deal with both convex and non-convex problems, unlike the weighted sum method. However, its implementation requires finding ideal vector $Z^\star$. This can be found by optimizing for every individual objective function. However, what is less ideal is the need to know the actual minimum and maximum objective values. These could be found by again optimizing with minimization and maximization for every objective function, but this is not ideal when the number of objectives increase, and the size of the objectives (the complexity of the process) increases. Lastly, this method is limited by the p-value, for a small p-value, not all Pareto-optimal solutions are obtained, and on the other hand, for large p-value the complexity of the objective function increases.

Comparing all three (3) methods, the weighted sum method is the first method to fall out of favor. It requires too much prior knowledge about the shape and behavior of the models and solution spaces for it to be a feasible technique to use in a generalistic optimization method. It is also not able to optimize for non-convex solution spaces. The other two (2) methods are both feasible candidates for multi-objective optimization techniques within this optimization problem. The primary advantage of the weighted metric approach over ε-constraint is a guarantee to find all Pareto optimum solutions. However, where the weighted metric method requires the actual absolute min and max values for all objective functions, a guarantee of finding all Pareto optimum solutions can also be achieved with the ε-constraint method with a less strict requirement. With the ε-constraint, a theoretical min and max value can also be used, which, in the optimization problem of this thesis, is relatively easy to obtain. Furthermore, it is only required for the constrained objectives, not for all. Lastly, the ε-constraint is a much simpler method to implement within the conditions of this thesis, whilst theoretically being able to achieve the same results. With this in mind, the multi-objective method which will be used in this thesis will be the **ε-CONSTRAINT METHOD**.

3.5.3 Implementation of the ε-constraint method

The ε-constraint method requires three (3) components: the primary objective function, the constrained objective function(s), and some epsilon value. The main element which needs to be figured out for implementation is setting the ε-constraint parameter for ***Equation 25***. As stated in **ε-Constraint** this parameter needs to be set carefully in order to prevent it from constraining the constrained objective outside of its feasible solution space. It should also be set in such a way that the objective is not compromised too much, thereby effectively turning the problem into a single-objective optimization problem.

One way of setting the right epsilon value, is to define it as a point between the minimum and maximum values of the objective function. However, since the shape and behaviour of the models and solution space are not known prior to running the optimization step, the real min and max values are not known. Thankfully though, it is possible to construct a theoretical min and max for the objective function with relative ease. Even though the shape and behaviour of the objective functions, and the solution space are not known, it is known how to calculate the objective function values using the functions from **Objective Functions**. And, a set of models, as regression models per activity-resource combination, is also available, and in generating the models, the mean performance of a particular activity-resource combination can be recorded. With these components, a theoretical min and max value for an objective function can be found.

In order to find Tmin and Tmax, where T indicates theoretical, the objective function needs to be calculated twice. Once with the most efficient (lowest mean), for that objective, activity-resource combinations, and once with the least efficient (highest mean), for that objective, activity-resource combinations. Effectively this done by finding all target activities, activities in the current variant to be optimized, and then for every activity find the best or worst resource based on the mean respectively. This should be done without any regard for the constraints set in **Constraints**. This also explains why these min and max values are theoretical. The activity-resource combination sets for min and max might not be feasible solutions because they might not meet the constraints. However, for defining the epsilon range, this is acceptable, because this still approximates the actual min and max values of the objective function. Adding to that, there is a fair probability that the theoretical min and max values are smaller and greater than the actual min and max values of that objective, thereby giving a fair probability that all solutions on the Pareto optimum front can be found by picking points for epsilon between the Tmin and Tmax values.

Concretely, the above translates into the following constraint setup:

$$\forall o \in O^{-}, c \in C, \epsilon \in \mathrm{E}: \epsilon_o = c_o(T^{+} - T^{-}) + T^{-}$$

Equation 27 – The ε-constraints worked out for this application, where ϵ_o from **Equation 25** *are defined as a point in the range between T^{+} (*Tmax*) and T^{-} (*Tmin*), where the point in this range is defined by c_o which a the compromise-allowance factor, between (inclusive) zero (0) and one (1) and is a real number, for a given objective from the larger set C which is the set of all compromise-allowance factors for all objectives in O^{-}. O^{-} is the set of all objectives, with the exception of the primary target objective and o is an objective from this set, and finally E is the set of all constraining epsilon parameters respective to the set of objectives and ϵ is an epsilon constraint from this set.*

Note that there is an extra factor added in this constraint setup: c. This factor is the compromise-allowance, which is a value between (inclusive) zero (0) and one (1). This value indicates how much the constrained objective is allowed to be compromised on. This essentially indicates how close to Tmin or Tmax the constrained objective should be. Note, the compromise-allowance $c \neq \epsilon$! It does however decide the final epsilon value.

A little back, it was mentioned that finding all target activities is related to the current variant. This is because, just like with single objective optimization, multi-objective optimization, or optimization as a

whole in order to improve the performance of a process, is not done on the entire process, but on its variants as is described in **Perform variant optimizations**.

Applying the Tmin and Tmax technique of finding the best and worst activity-resource combinations to the solution space example from ***Figure 11***, the following new solution space is developed:

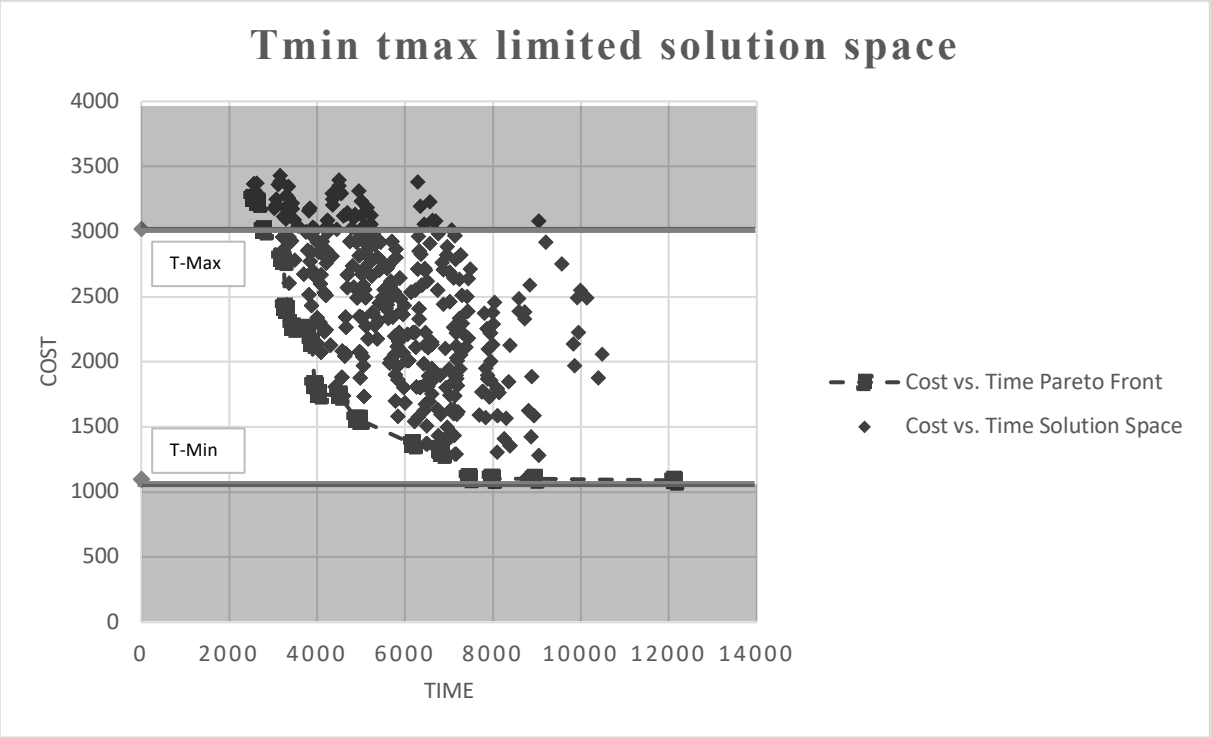


*Figure 12 – Pareto Optimum front demonstration for the process in **Figure 9**. With an added Tmin Tmax solution space constraint for the cost objective. Please refer to **10.9** for a larger version.*

As described earlier, Tmin and Tmax could theoretically be smaller and/or greater than the actual min and max of its respective objective function, but, as is seen in this case, this is not always the case. However, despite not all solutions being available, still a large set of solutions along the Pareto optimum front are available.

The actual constrained solution space can now be found by applying a compromise-allowance factor. In this case, Tmin and Tmax are ~1.102 and ~3.024 respectively. If an arbitrary compromise-allowance factor of for example 0,35 is applied, meaning that the calculated value for the cost objective may not be worse (higher) than 35% of the Tmin and Tmax constrained solution space, then the ε-constraint evaluates to ~1.775 in accordance with ***Equation 27***. This then results in the following ε-constrained solution space:

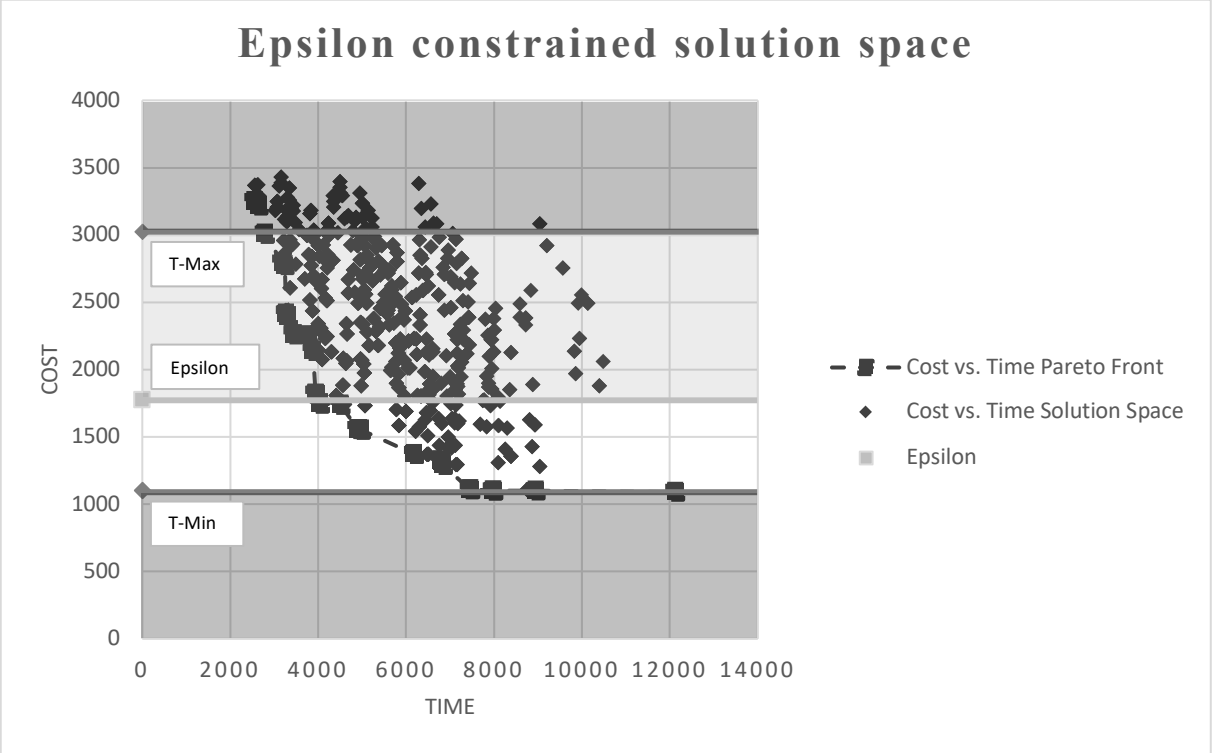


*Figure 13 – Pareto Optimum front demonstration for the process in **Figure 9**. With an added Tmin Tmax solution space constraint and an applied ε-constraint with a compromise allowance of $c = 0.35$ for the cost objective. Please refer to **10.10** for a larger version.*

This leaves only the unmarked area as the feasible and allowed solution space. Next, the optimizer will optimize, rather minimize, for the throughput time objective. The solution with lowest possible throughput time will most probably evaluate to the solution of: $\{o_t: 4.046{,}36; o_c: 1.748{,}91\}$. This is worse than the best solution for either throughput time or cost in single-objective optimization, but it could be an acceptable compromise between the two (2) objectives where neither result would increase to an extreme value.

4 Data Requirements

This section about data requirements will cover two (2) aspects on data, namely the shape and format of the data required to perform optimizations. The first part is the requirements for the initial process mining part, where a process tree is constructed based on trace data, and the second part is about the data requirements for the model construction step discussed in **Find resource cost figures** and **Cost Figure Variants**. Besides the shape and format of the data, the quantity aspect will also be discussed in this section. The quantity question is split in two (2) parts, one (1) specifically aimed at the quantity requirements for the process miner, and the second part is about the requirements for the cost modelling techniques.

4.1 Optimizer data requirement

The data requirements from the process optimizer is split up in three (3) parts. The requirements from the process miner (the inductive miner), the requirements from the optimizer itself and then the requirements from the different objective functions. The three (3) subsections below cover these three (3) parts.

4.1.1 Process miner data requirements

The process miner used for the optimizer in this thesis is the pm4py process miner [10], [11]. This process miner has three (3) specific data requirements. From the pm4py process mining library, the inductive miner is used (refer to **Existing theories on process mining** for an in-depth elaboration on the miner choice), which tries to construct a chronologically correct process tree, based on a data trace. The inductive miner has three (3) data requirements, which are as follows:

- **CASE IDENTIFIER** every trace must have case identifiers which are unique and assigned to one (1) single process instance;
- **ACTIVITY IDENTIFIERS** activities in a process must have an identifier, unique number or unique descriptor string. Activity ids are unique and consistent across all cases. Activities may occur multiple times in a process;
- **START TIMESTAMP** the start timestamp is used to indicate the order of activities in a process, indicate loop order (distinguish between base and looping activities), and show concurrent activity occurrences;

As stated above, time stamps help determine which activities happen concurrently, or rather, in parallel. There is one (1) special condition to detect whether activities, or sub processes, occur in parallel. The process miner only picks up a parallel occurrence when the sub processes are repeatedly shown in parallel, and only when their starting order varies between cases. This phenomenon can be demonstrated with the following two (2) example cases:

Table 12 – Example process for parallel activity occurrence.

Case ID	Activity ID	Start Timestamp
0	a	00:00
0	b	01:00
0	c	01:00
0	d	03:00
1	a	15:00
1	b	16:30
1	c	16:30
1	d	20:00

In this trace activities *c* and *b* occur in parallel. However, when applying the inductive miner, all activities will be shown in sequence. An output of the process miner for this example dataset can be seen in ***Figure 14***.

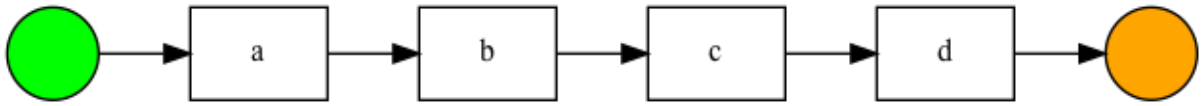


Figure 14 – BPMN process representation of the example process shown in **Table 12**.

Table 13 – Example process for parallel activity occurrence.

Case ID	Activity ID	Start Timestamp
0	a	06:00
0	b	07:00
0	c	07:00
0	d	10:00
1	a	12:00
1	c	13:00
1	b	13:00
1	d	16:00

This second trace has a similar setup as the first trace, where again activities *c* and *b* occur in parallel. But this time activities *c* and *b* have been swapped around between the two (2) cases, whilst having identical start timestamps. Even though the two (2) processes are identical, with exception to the activity trace recording swap, the second trace will show a parallel sub process in the inductive process discovery result. An output of the process miner for this example dataset can be seen in ***Figure 15***.

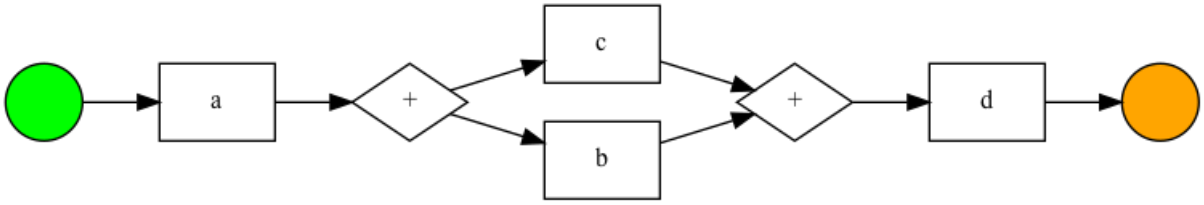


Figure 15 – BPMN process representation of the example process shown in **Table 13**.

4.1.2 Process optimizer base data requirements

Besides the requirements from the process miner, there is one (1) other base requirement. The optimizer in this thesis attempts to optimize the allocation of resources to activities. The fourth requirement, coming from the optimizer, is therefor *resource ids*. Concretely the resource identifier requirement is as follows:

RESOURCE IDENTIFIERS resources get unique identifiers, unique numbers or unique descriptor strings. Resource ids must be consistent across all cases. Resources can occur multiple times in a case and can be assigned to multiple activities and/or an activity multiple times. Multiple resources may not be assigned to one (1) instance of an activity at the same time, but they can be assigned to multiple instances of an activity in a case.

4.1.3 Objective function data requirements

In this thesis, as is introduced in **Optimization definition**, there are three (3) objective functions. These three (3) objectives bring forth their own data requirements. This thesis will only cover the additional data requirements for the three (3) objective functions discussed in this thesis. However, with the introduction of more objective functions, more data format requirements can arise.

The data requirement for the three (3) objectives are as follows:

- **ACTIVITY EXECUTION DURATION TIME** This field indicates how long the activity-resource combination took to finish its task. The format or unit may be arbitrary, as long as it is consistent

between cases. The optimizer will not calculate the execution duration time on its own based on timestamps, as this might be ambiguous;

- **ACTIVITY WAIT TIME** This fields indicates how long this activity-resource combination had to wait until it could start its task. The format or unit may be arbitrary, as long as it is consistent between cases. The optimizer will not calculate the execution duration time on its own based on timestamps, as this might be ambiguous;
- *(Optional)* **END TIMESTAMP** This optional data column could be added in order to calculate the activity execution duration time and/or wait-time in pre-processing if this is absent prior to the start of the optimization process;
- **MONETARY COST** This indicates how much the execution of a task in an activity-resource combination cost. The unit may be arbitrary, as long as it is consistent between all cases.

4.2 Data quantity requirements

The second question related to data requirements is: how much data is required at a minimum. There are two (2) components which should be considered when trying to answer this question. Both the process miner and the cost models have their own data quantity requirements.

4.2.1 Process miner data quantity requirements

Process mining does not necessarily have a minimum requirement for discovering processes [49]. Sequential activities and loops could be discovered within one (1) or two (2) cases, as long as the activities are unique in their identification. For parallel components an anecdotal example was provided in Fout! Verwijzingsbron niet gevonden., showing that in order to discover these two (2) parallel branches, two (2) cases should at least be present where the two (2) branches are swapped in occurrence. One (1) hypothesis which could arise is then that the minimum number of required cases (in which the branches are all swapped) is determined by the number of parallel branches in a parallel component, because they need to occur in several different configurations.

Another element worth noting when discussing process miners, is the XOR-choice component. In a process-case only one (1) branch of a XOR-choice component will be shown at a time. So, in order to cover all different choices, the number of cases should hypothetically be the number of different choice branches for an XOR-choice component.

More importantly than covering all different tree variants, is having a representative dataset. If certain variants occur 30% of the time when running an instance of a process, then the dataset should closely mirror that. The number of required cases, or rather the quantity requirement for process mining therefor is hard to define, it should at least be representative, and the exact requirement to cover a large majority of the different variants, completely differs depending on the complexity of a process tree.

4.2.2 Objective function modeller data quantity requirements

The objective function modeller in this thesis is based on either linear or polynomial regression. Note that there is not one (1) big model, but rather multiple smaller models, therefor this minimum requirement is on a per model basis.

Regression depends on an intercept and a (set of) slope(s) as is shown in **Cost Figure Variants**. Linear regression, by virtue of its inner workings, could determine the intercept and slope of a dataset with two (2) datapoints [36], but this would probably not give a good correlation between the variables. There are statistical tests which could help determine whether the number of datapoints would result in a representative model, one of these tests would for example be a *Type two (2)* risk error test. However, there have been done several studies which ask this exact question [50], [51]. From these studies, the answers range from between ten (10) datapoints per variable in a model, to at least 25 datapoints to get a representative model.

5 Experiment Setup

In the method and design section (**Method & Design**) the process optimizer has been fully conceptualized and designed. However, several parts of the process optimizer method have several possible variants which can be used to achieve the same goal. For these components of the process optimizer, it has yet to be determined which method is the overall best method for that particular component. Figuring out which method is best per component, will be done through a set of experiments.

A second objective in the experiments is, is to show whether the optimizer can improve processes or not and how it behaves depending on changing parameters and processes. This is not expected to give an absolute improvement potential promise, but rather show that the divide and conquer concept is functional.

This section will be split up in three (3) parts. First the setup of the datasets for the experiments are discussed. Secondly the experiments themselves will be set up. Finally, the third section is about the experimentation environment on which all data is generated, and on which all experiments are conducted.

Note that all experiment data, intermediary results, configurations and prototype source codes are available through the repository linked with this thesis. The prototype source codes includes both the source codes for the optimizer as well as for the experiment data generator. The repository is accessible through *"https://github.com/MarcelKolen/process-mining-resource-allocation-optimizer"*.

5.1 Experiment Data

To cover a wide set of different scenarios, several different process trees, with varying properties, will be generated. For each trace all activities in the different trees receive a randomly generated model which will follow a set of rules. And, for every trace related to a tree, a set of resources will be added which are also randomly generated but do follow a set of rules and constraints.

These datasets will be generated with a five (5) step experiment-trace generator. The five (5) steps are as follows:

1. **PROCESS TREE CONSTRUCTION** The first step is to randomly generate a process tree with a set of unique activities;
2. **ADD SIMULATION VALUES TO PROCESS TREE** To allow for different variants, the tree has to be extended with simulation values;
3. **CONSTRUCT SIMULATION ACTIVITIES** Next is the construction of simulation values for the activities, where certain base values and modifiers for the objective functions are added to the activities;
4. **CONSTRUCT SIMULATION RESOURCES** In order to make the datasets suitable for activity-resource allocation optimization, resources also have to be added which will also have base values and modifiers for the objective functions;
5. **GENERATE TRACES** Lastly a number of simulation runs will be performed based on the rules and values set out in the previous steps in order to construct a dataset with multiple cases.

This section will cover how the experiment-trace generator works as well as how the parameters are setup in order to achieve a diverse set of experiment datasets.

5.1.1 Process trees

A set of five (5) different process trees will be generated. Three (3) trees will consist solely out of one of the primary components, those being: loops, parallel sub-processes and XOR-sub-processes as introduced in **Determine process variants**. Two (2) other trees will be combined trees, where both trees

should contain all tree components, totalling five (5) trees which are to be used to construct experiment data.

Note that for the following five (5) trees, sequential structures are always included, as trees often only exist when they have sequential activity structures.

These trees are going to be generated using a pre-existing tree generating algorithm from the pm4py library. Pm4py has several process simulation algorithms [10], [11], however in this thesis only a simple tree generator will be used. This tree generator does not produce traces, it only produces a process tree.

This process tree generator takes a set of parameters. The list of (default) parameters (settings) can be found on the pm4py reference page [11]. For every process tree below, the used parameters and generation seeds will be recorded. There is one (1) parameter which deviates from the default settings for all trees, which is the *silent probability* parameter. This will be set to zero (0) for all trees.

5.1.1.1 Loops only

This tree will only include loops as a special tree structure. Parallel and XOR-choice sub-processes are not included in this tree type. The parameters used in the process trace generator for the tree generator are as follows:

Table 14 – Settings used for the loops only tree.

Seed 0	Seed 1	Min	Max	Mode	Silent (%)	Choice (%)	Parallel (%)	Loop (%)
4674	6666	5	8	6	0.0	0.0	0.0	25.0

A BPMN representation of the tree is as follows:

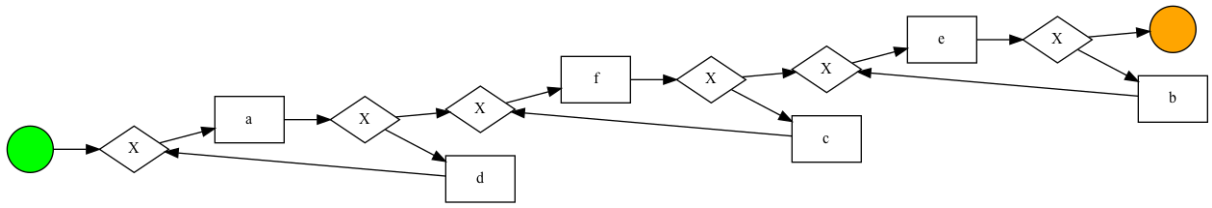


Figure 16 – Loops only tree based on the settings shown in **Table 14**.

The loops only tree has been added because one (1) of the aspects of model generating, as described in **Find resource cost figures** and **Cost Figure Variants**, is tracking resource and activity occurrence. The current cost description models in this thesis also have two (2) input variables: resource and activity occurrence for a respective activity-resource pair. This tree type should specifically stress the model component of the process optimizer.

5.1.1.2 Parallel branches only

This tree will only include parallel. Loops and XOR-choice sub-processes are omitted from this tree type. The parameters used in the process trace generator for the tree generator are as follows:

Table 15 – Settings used for the parallel branches only tree.

Seed 0	Seed 1	Min	Max	Mode	Silent (%)	Choice (%)	Parallel (%)	Loop (%)
722101364	1020699242	5	8	6	0.0	0.0	25.0	0.0

A BPMN representation of the tree is as follows:

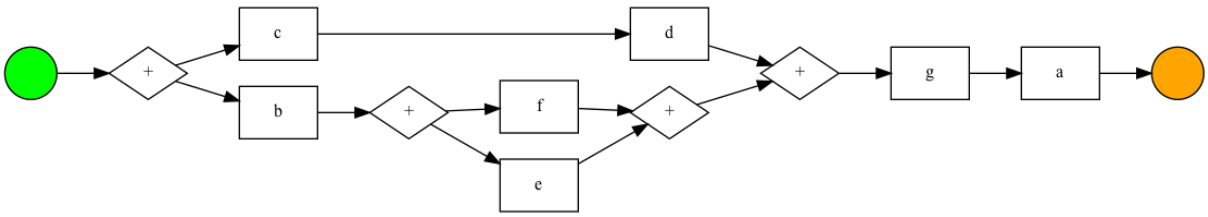


Figure 17 – Parallel branches only tree based on the settings shown in **Table 15**.

The parallel branches only tree has been added because one (1) of the objective functions, the time objective function specifically, needs to correctly deal with parallel occurrences of activities as described in **Objective Functions**.

5.1.1.3 XOR-branches only

The next tree will only include XOR-choice sub-processes. Loops and parallel sub-processes are omitted from this tree type. The parameters used in the process trace generator for the tree generator are as follows:

Table 16 – Settings used for the XOR-branches only tree.

Seed 0	Seed 1	Min	Max	Mode	Silent (%)	Choice (%)	Parallel (%)	Loop (%)
1911086572	193421582	5	8	6	0.0	25.0	0.0	0.0

A BPMN representation of the tree is as follows:

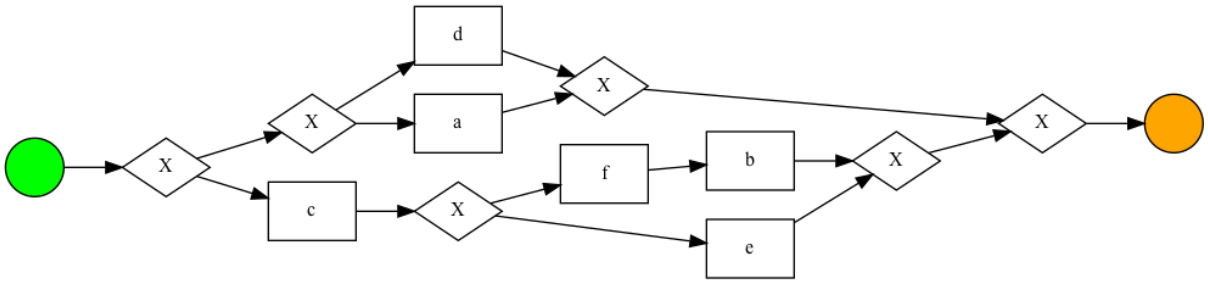


Figure 18 – XOR-branches only tree based on the settings shown in **Table 16**.

The XOR-choice sub-processes tree has been added because one (1) of the aspects of the process optimizer is to detect process variants as is described in **Determine process variants**. This tree should specifically stress the process variant search component of this process optimizer.

5.1.1.4 Small, combined tree

The second to last tree will be a combined tree with loops, parallel sub-processes and XOR-choice sub-processes. The parameters used in the process trace generator for the tree generator are as follows:

Table 17 – Settings used for the small, combined tree.

Seed 0	Seed 1	Min	Max	Mode	Silent (%)	Choice (%)	Parallel (%)	Loop (%)
775451721	548953530	5	8	6	0.0	25.0	25.0	25.0

A BPMN representation of the tree is as follows:

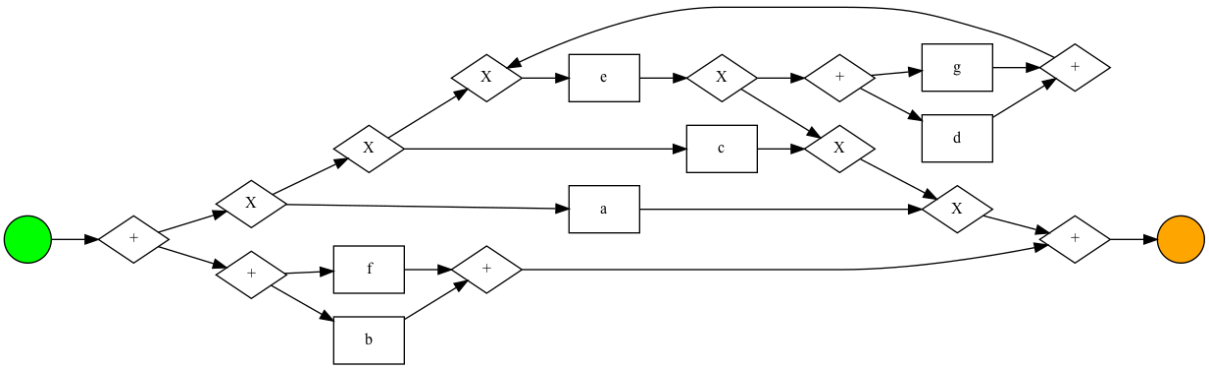


Figure 19 – Small combined tree based on the settings shown in **Table 17**.

The small, combined tree is set up to stress all components of the process optimizer at once.

5.1.1.5 Large, combined tree

Finally, the last tree will be a large, combined tree with loops, parallel- and XOR-choice sub-processes. The parameters used in the process trace generator for the tree generator are as follows:

Table 18 – Settings used for the large, combined tree.

Seed 0	Seed 1	Min	Max	Mode	Silent (%)	Choice (%)	Parallel (%)	Loop (%)
1961566938	1793279626	8	16	12	0.0	25.0	25.0	25.0

A BPMN representation of the tree is as follows:

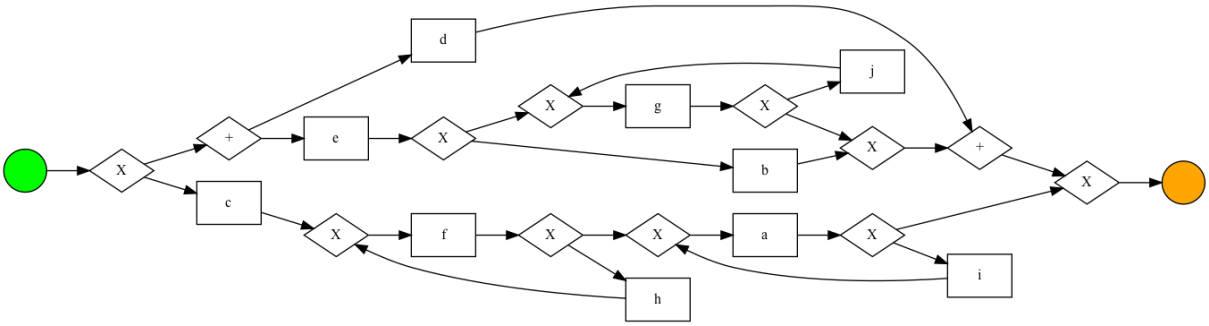


Figure 20 – Large combined tree based on the settings shown in **Table 18**.

The small, combined tree is set up to stress all components of the process optimizer at once.

5.1.2 Simulation values

In order to generate traces, the trees are going to need *simulation values* attached to their components. These simulation values dictate for example how often, and with what probability a loop may occur, and they dictate the probability distributed across XOR-choice sub-process children.

For each tree, three (3) different simulation settings will be generated. The seed combinations are recorded for every tree type. The simulation value generator takes three (3) parameters. The three parameters, together with the base settings for each tree are as follows:

Table 19 – Base settings for generating all tree simulation settings.

Max loop depth	Decrease loop probability	Even XOR probability
6	Yes	No

The settings in the table above are explained in Fout! Verwijzingsbron niet gevonden..

The simulation values and seeds for every tree are described in **10.11**. In the following three (3) subsections the two (2) simulation elements will be explained as well as their resulting formats.

5.1.2.1 Loop simulation values

The loop component of the trace simulator has two (2) "parameters" which can be set. The first parameter is the max loop depth, and the second is the probability of a loop initiating another instance of itself.

Every loop in the process tree will get a *list of instance probabilities*. This list has two (2) purposes. The first purpose is to indicate how often a loop may repeat. The second purpose of the *list of instance probabilities* is to indicate what the probability of a loop starting another instance of itself is. This list works as follows: Take the list of probabilities shown in ***Table 20***.

Table 20 – Example loop instance probability list.

Instances	0	1	2	3	4
Probabilities	86%	74%	55%	31%	17%

This list has five (5) probabilities, for five (5) instances. This means that the loop may only happen five (5) times. If this first loop occurs, then the process simulator will look at the second instance. This second instance has probability of 74% of occurring. This continues until all values have been covered.

The first of the two (2) parameters, *max loop depth*, dictates the maximum number of loop instances. Note that the number of loops instances, which is randomly determined, can be less than the maximum.

The second parameter, *decrease loop probability*, dictates how the probabilities are set. If this parameter is set to *true,* then the probability of a next instance may not be higher than its predecessor unless it is the first instance. If this parameter is set to *false* then each loop instance can have a probability between zero (0, or 0%) and one (1, or 100%). The domain of the probability of every instance can essentially be defined as follows:

$$p \subseteq [0,1], \delta \in \{\text{true,false}\}: D_{p_{i,\delta}} = \begin{cases} [0,1], & i = 0 \vee \delta = \text{false} \\ [0, D^{+}_{p_{i-1,\delta}}], & i > 0 \wedge \delta = \text{true} \end{cases}$$

Equation 28 – Probability domain D_p for every instance i of a loop probability list where δ indicates whether "decrease loop probability" is active or not, and $D^{+}_{p_{i-1,\delta}}$ indicates the upper bound of a probability domain of instance i.

5.1.2.2 XOR-Choice simulation values

XOR-choice components are, dependent on probability settings. XOR-choice sub-trees have a set of two (2) or more children, and each of these children has a probability of occurrence.

The behaviour of the formation of this list of child probabilities depends on the earlier mentioned parameter *XOR even probability*. If this parameter is set to *true* every child in an XOR-choice sub-tree will have an equal occurrence probability. If the *XOR even probability* parameter is set to *false* then every child will have its own probability, possibly different from the others, as long as the sum of probabilities is equal to one (1 or 100%).

5.1.2.3 Simulation tree format

The simulation values are added in a specific format. This format is setup as follows:

SEQUENTIAL COMPONENT The sequential component consists of the sequential operator ('>') as the first element, and a list of children as the second element to the tuple:

```
('>', [list of children])
```

Figure 21 – Sequential component of a process tree with '>' as its operator.

PARALLEL COMPONENT The parallel component consists of the parallel operator ('+') and a list of children, or rather, a list of parallel sub trees:

```
('+', [list of children])
```

Figure 22 – Parallel component of a process tree with '+' as its operator.

LOOP COMPONENT The loop component operator is '*'. The loop operator is followed by a list of loop instance probabilities. The third element is the list of children. Note that a loop only has two (2) children in the process trace generator described in this thesis. The first child is the base activity, and the second child is always the looping activity or looping sub-tree. The loop component is represented as follows:

```
('*',
 [list of loop instance probabilities],
 [list of children])
```

Figure 23 – Loop component of a process tree with '' as its operator.*

XOR-CHOICE COMPONENT The XOR-choice component has three (3) elements. The XOR-choice operator ('X'), a list of child instance probabilities and a list of children. Note that the position of a probability in the list corresponds with a child in the same index position in the list of children. The XOR-choice component is represented as follows:

```
('X',
 [list of child occurrence probabilities],
 [list of children])
```

Figure 24 – XOR-choice component of a process tree with 'X' as its operator.

5.1.3 Activity simulation values

The third step is to add simulation values for each objective function (time and cost in this case) to the activities. All simulation values for the activities for all trees are generated with the same base parameters. The random seeds used for each tree are the same as the seeds as the ones in **Simulation values 10.11** . Note that for each tree, three (3) simulation setups are constructed. And for every simulation setup, a unique set of activity settings is setup.

The base parameters give a probability range for how much the activity "costs" when executed for each objective. These are set as follows:

Table 21 – Base settings for generating all activities.

Min Time	Max Time	Min Cost	Max Cost
10	1000	10	1000

In order to simulate re-occurrence effects, for example with loops, the activities will receive a modifier function. The independent variable indicates how often an activity is repeated in a case. Multiple modifier functions can be provided to the experiment-trace generator, but only one (1) will be assigned to an activity. This happens randomly according to a given probability distribution. For the re-occurrence functions, the following settings are used:

Table 22 – Modifier functions with re-occurrence of an activity (x) as input.

Time functions	
Function	**Probability distribution**
$f_{A_t}(x) = \frac{1}{2x} - \frac{1}{2}$	$\frac{2}{3}$
$f_{A_t}(x) = \sqrt[4]{x}$	$\frac{1}{3}$
Cost functions	

Function	Probability distribution
$f_{A_c}(x) = -\frac{x\sqrt[4]{x}}{20}$	$\frac{4}{5}$
$f_{A_c}(x) = -\frac{3x\sqrt{x}}{20}$	$\frac{1}{5}$

These functions are justified in the following two (2) sub-sections.

5.1.3.1 Throughput time functions

There are two (2) throughput time functions which can be used. The first function is a linear asymptotic function with a maximum value of zero (0) and a minimum value of $\lim_{y \to -\frac{1}{2}} y$ (approaching -0.5) within the permissible domain D_x: $[1, \infty)$. This function should decrease the throughput time "cost" with a diminishing and limited return. This could for example simulate a production activity: when more items are produced, the production efficiency increases and therefor the throughput time for every production cycle decreases.

The second function is root function with a maximum value of $\lim_{y \to \infty} y$ (approaching infinity) and a minimum value of one (1) within the permissible domain D_x: $[1, \infty)$. This function is designed to increase the throughput time of an activity gradually when it is repeated. This function could simulate for example an activity such as a repair service activity. Where when an activity has to be repeated, the original problem has not been alleviated yet and the complexity of the repair activity has to increase.

5.1.3.2 Cost functions

The cost functions are both negative root functions one with a maximum value of $-\frac{1}{20}$ and the other with a maximum value of $-\frac{3}{20}$ and both with a minimum value of $\lim_{y \to -\infty} y$. One (1) function has a slower cost decrease whereas the other function has more rapid cost decrease per activity run. Both are based on the idea that starting an activity initially brings the highest costs, but once this activity has been established, the cost per run of an activity decreases as the initial "investments" have already been done.

Finally, the exact values for all activities for every tree, and their subsequent simulation variants, are described in **10.12**.

5.1.4 Resource simulation values

The last components which are needed in order to generate experiment trace data, is a set of resource to randomly assign, with a probability distribution, to the activities. In order to cover a wider set of scenarios, four (4) different sets of activities are created for each tree. As with the activities setup and simulation setting setup for the trees, for all four (4) sets will be generated three (3) times with differing seeds. The seeds are the same seeds as used in the previous sections.

The four (4) sets of resources result from two (2) properties which can be applied. These properties are: a set of resources will have both average performers and specialized performers, or only average performers; and, the average resources will either have no random effects added to them, or they will have random effects added to them. A subset of the average resources is limited to a randomly selected few activities, so not every resource from this subset can perform all activities. Another set is freely allocatable. The freely allocatable average resources receive a *jack of all trades penalty*. The specialized resources are always unrestricted, and they also do not suffer any jack of all trades penalties.

The random effects are only added to the throughput time performance of a resource, and they will only be applied to average performers, not to the specialized resources. These random effects should simulate occurrences such as strikes, large holdups/traffic jams due to protests, heavy weather, etc. What should be taken into account however is that these random effects are not at all balanced or in scale related to aforementioned examples, they are merely randomly assigned modifiers.

The resources have a set of base settings. There are four (4) sets of settings to be distinguished: one (1) for average resources without random throughput time modifiers, one (1) for average resources with random throughput time modifiers, one (1) for throughput time effective resources, and finally one (1) for cost effective resources. The subsections below describe the setup of the four (4) resource groups.

5.1.4.1 Base settings: average resources without random throughput time modifiers

The **AVERAGE RESOURCES WITHOUT RANDOM THROUGHPUT TIME MODIFIERS** resource generator receive the following base settings:

Table 23 – Base parameter settings for average resources without random modifiers.

Min number of resources	Max number of resources[1]	Jack of all trades penalty	Min time base modifier	Max time base modifier	Min cost base modifier	Max cost base modifier
3	10	0.25	-0.3	0.3	-0.2	0.2

In order to simulate reallocation effects, the resources will receive a modifier function. The independent variable indicates how often an activity is repeated in a case. Multiple modifier functions can be provided to the experiment-trace generator, but only one (1) will be assigned to an activity. This happens randomly according to a given probability distribution. For every function, a threshold parameter is added. This is a randomly determined value within a given threshold range. This threshold parameter determines a crossover-point for the functions below where they shift behaviour. For the re-occurrence functions, the following settings are used:

Table 24 – Modifier functions with allocation of a resource x and a threshold value τ as input for average resources without random modifiers.

Time functions	
Function	**Threshold range**
$f_{R_t}(x,\tau) = \begin{cases} \frac{x}{2}, & x \leq \tau \\ (x-\tau)^2 + \frac{\tau}{2}, & x > \tau \end{cases}$	D_τ: [3,7]
Cost functions	
Function	**Threshold range**
$f_{R_c}(x,\tau) = \begin{cases} -\frac{\sqrt{x}}{10}, & x \leq \tau \\ -\frac{\sqrt{\tau}}{10}, & x > \tau \end{cases}$	D_τ: [5,10]

The throughput time function is based on the inverted U-theory, also known as the Yerkes-Dodson Law [25] and the findings of other related, in terms of task-performance-task-pressure-relation, studies. Several studies state that employee performance has some inverted U-shape relation to task pressure and/or a relation of lower performance with (too) high work pressure [26], [27]. The essence of this theory comes down to the following: if an employee is not pressured enough, then their performance will be sub-optimal. This sub-optimality of their performance could for example be due to boredom. On the other hand, if an employee is pushed too hard, so if the task pressure is too high, then the performance of this employee will falter as well due to them becoming strained, stressed, or even in worst-case scenarios, burned-out.

In this experiment setup, the throughput time modifier function indicates a level of performance. Pressure will be defined as the number of activities a resource has to perform. The function is based on a simplified version of the U-theory. In this experiment, the resources are set to already perform at an

[1] Due to a bug within the trace generator, if resources are set to be limited to either some activities, or not be limited at all, there will always be the maximum number of resources. When all resources should be limited to a set of activities, then the number of resources is always the minimum (or close to the minimum) allowed amount. For all settings: please assume that the number of resources will always be the max amount!

optimum, so there is no minimum requirement to perform optimally. Instead, the resource will only be modelled to suffer from higher stress as their task pressure increases. At a certain level of pressure, which is determined by a random value τ in order to simulate stress-tolerance variance between resource, the performance of a resource will decrease in a quadratic fashion. This simulates a resource being burned out.

The cost function is based on the notion that allocating a resource pays for that resource's time, and that when a resource is allocated multiple times, the cost of that resource can be shared between allocations. This is however set to be based on diminishing returns with a hard cap. This hard cap varies between resources and is determined randomly by the value τ.

5.1.4.2 Base settings: average resources with random throughput time modifiers

The **AVERAGE RESOURCES WITH RANDOM THROUGHPUT TIME MODIFIERS** resource generator has the same base settings and parameters as the resources without a random throughput time modifiers. The only difference is between the throughput time cost function, which now has a random element added to it:

Table 25 – Modifier function for throughput time with allocation of a resource x and a threshold value τ as input for average resources with random modifiers.

Time functions		
Function	**Threshold range**	**Random value range**
$f_{R_t}(x,\tau) = \begin{cases} \frac{x}{2} + \xi, & x \leq \tau \\ (x-\tau)^2 + \frac{\tau}{2} + \xi, & x > \tau \end{cases}$	D_τ: [3,7]	D_ξ: [0.01,0.1]

This throughput time function is similarly setup as the one in the section above. The principles are also based on a simplified version of the inverted U-theory. The only difference is the addition of a random element which has been explained in the overall resource setup description.

5.1.4.3 Base settings: throughput time specialized resources

The **THROUGHPUT TIME SPECIALIZED RESOURCES** resource generator receives the following base settings:

Table 26 – Base parameter settings for throughput time specialized resources.

Min number of resources	**Max number of resources**	**Min time base modifier**	**Max time base modifier**	**Min cost base modifier**	**Max cost base modifier**
2	4	-0.5	-0.5	0.5	0.5

The throughput time specialized resources have the following functions applied to them.

Table 27 – Modifier function for throughput time and cost with allocation of a resource x as input for throughput time specialized resources.

Time function	**Cost functions**
$f_{R_t}(x) = \frac{\sqrt{x}}{10}$	$f_{R_c}(x) = -\frac{\sqrt{x}}{5}$

Just like with the average performer resources, the time function for the specialized resources is based on an idea that as resources are pressured more (beyond their optimal pressure point), their performance will decrease. However, the throughput time specialized resources are much less affected by an increase of pressure than average resources. As a trade-off, the cost function does not decrease as significantly as compared to the average performing resources, with the idea being that these resources are higher performing (for longer) but therefor also more expensive to allocate.

An example of these resources could for example be work-field expert working as consultants. They are flown-in resources which bring more expertise and therefor a higher performance than the average resources, but they are more costly due to their fly-in nature.

5.1.4.4 Base settings: cost specialized resources

The **COST SPECIALIZED RESOURCES** resource generator receives the following base settings:

Table 28 – Base parameter settings for cost specialized resources.

Min number of resources	Max number of resources	Min time base modifier	Max time base modifier	Min cost base modifier	Max cost base modifier
2	4	0.5	0.5	-0.5	-0.5

The cost specialized resources have the following functions applied to them.

Table 29 – Modifier function for throughput time and cost with allocation of a resource x as input for cost specialized resources.

Time function	Cost functions
$\boldsymbol{f_{R_t}(x) = \frac{x^2}{10}}$	$f_{R_c}(x) = -\frac{\sqrt{x}}{100}$

These resources are designed to have minimal cost, which is reflected by their base settings and cost function. However, this is at a cost of large throughput time penalties both in the base settings and in the time function. The throughput time function models low pressure resistance.

An example of these resources could be (young) intern employees. An intern employee could be employed at a (much) lower cost as compared to average employees, but due to their inexperience, they could perform worse at their given tasks, and they could possibly be overworked earlier as compared to average employees.

5.1.5 Case counts & Dataset Combinations

For all combinations of the trees, the simulation values of the trees, and resource sets variants, except for the combinations with the large, combined tree, the number of cases (simulation runs) is 2,000. All runs with the large, combined tree will receive 5,000 cases (simulation runs). This makes for five (5) times three (3) times four (4) combinations, which results in 60 different datasets. 48 of these datasets have 2,000 cases, and 12 datasets have 5,000 cases, for a total of 156,000 cases across the 60 datasets.

Ideally the number of cases covers a large enough set of variants, or at least a set which represents the most likely variant outcome. If an exhaustive coverage is to be reached, then the expected value of the negative binomial could be used to approximate the required number of cases.

$$\mathbb{E}(X) = \frac{n}{p}$$

Equation 29 – Expected value of the negative binomial where X is the binomial of n and p and where n is the number of times we want the event with probability p to occur.

Using the example from **Loop simulation values** where the last instance has a accumulated probability of $p = 1.8\%$ then if a full coverage is to be reached with $n = 1$, 56 cases would be required.

For every simulation case, three (3) steps are performed to go from the simulation settings to a process trace. These steps are explained in three (3) subsections below.

5.1.5.1 Play out the simulation tree

The simulation tree has a set of tree components, which are the sequential components, parallel components, looping components and XOR-choice components. For every case instance, the process trace generator randomly plays out the input simulation tree. This random playout is steered by the previously set probability values. In essence, there are only two (2) components which are developed into a played-out variant of themselves, the looping components and the XOR-choice components.

LOOPING COMPONENTS The looping components are pruned down into a sequential sub-tree with a pattern repeating child list consisting of a looping pattern of the base child and the looping child. How often a loop is initiated depends on the probability list set prior.

XOR-CHOICE COMPONENTS The XOR-choice components are also pruned down into a sequential sub-tree. The list of children of the sequential sub-tree will be the child branch of the XOR-choice component which is randomly chosen according to the set probabilities.

5.1.5.2 Attach resources

After the simulation tree has been played out into a variant for a case, the resources can be allocated to the activities. This is done by traversing down the entire simulation tree and finding every activity. For every activity which has been found, an allocation operation is commenced.

The allocation of resources to activities is done at random with a probability guide. The allocation operation works as follows:

1. First the subset of all resources which can be allocated is to be determined:
    a. In order to determine this subset, the first step is to find all unrestricted resources. These are resources are freely allocatable to any activity, and they are all automatically added to the allocation candidate list;
    b. Then, for the remaining resources, their restriction list has to be checked. If the current considered activity is found in the allocation restriction list of a resource, then this resource may be added to the allocation candidate list;
2. Once the allocation candidate list is determined, their allocation probabilities need to be determined:
    a. If during the construction of all resources, the parameter *all resources have an equal probability* is set to *true*, then the resource to be assigned is simply randomly chosen;
    b. If during the construction of all resources, the parameter *all resources have an equal probability* is set to *false*, then the allocation process first must determine the actual allocation probability of a resource. In order to determine the actual allocation probability of a resource, their probability base value has to be divided by the sum of all probability base values for the entire resource candidate list.

5.1.5.3 Generate traces

In the played-out simulation tree, there are now only two (2) types of operators, the sequence operator and the parallel operator. XOR-choice child choosing, and loop unrolling has been done before. This makes generating a trace a straightforward task. The trace generator will traverse through the entire tree, starting at the first operator, and traversing through its children, from left most in the list to right most in the list. This process is done recursively.

For every activity-resource the objective simulation results have to be calculated. The processing of these objective simulation results is the same for every objective in the current implementation. The following equation shows how this simulation result will be calculated, in this case for throughput time, however this can also be applied in the same way for the cost objective.

$$a \in A, r \in R: S_T(a,r) = T_b(a) \cdot \left(T_{f_A}\left(a, C_A(a)\right) + T_m(r) + T_{f_R}\left(r, C_R(r)\right)\right)$$

Equation 30 – Simulation result S_T for the throughput time where A is the set of all activities and a is an activity in this set and R is the set of all resources and r is a resource in this set. $T_b(x)$ is the base throughput time value for activity x, $T_{f_A}\left(x, C_A(x)\right)$ is the throughput time modifier function for activity x and $C_A(x)$ is the occurrence count of activity x in the current process variant. $T_m(y)$ is the base throughput time modifier associated to resource y and $T_{f_R}\left(y, C_R(y)\right)$ is the throughput time modifier function for resources y and $C_R(y)$ is the allocation count of resource y.

The second thing which needs to be done is to properly record, in correct format and order, the results from the simulations. The required format is described in Fout! Verwijzingsbron niet gevonden.. One (1) important part to consider is how to develop the start timestamp. The first activity can simply use some arbitrary initial timestamp. Succeeding activities will use a start timestamp which is calculated by taking the initial timestamp of the preceding activity and add the calculated simulation throughput time

of the preceding activity to that timestamp. When the preceding component is part of a parallel sub-tree the start timestamp is determined by taking the maximum of all the last activities their initial timestamp added together with their simulated throughput time in every parallel branch.

Parallel components throw one (1) last gotcha. Referring to Fout! Verwijzingsbron niet gevonden., the results of all parallel branches cannot simply be added in sequence. If the recursive results of the different children of parallel branches are simply recorded in order from left to right, as would be the case with sequential components, then the inductive miner might not be able to pick up on parallel occurrences. To combat this, the outcomes of the parallel branches are recorded separately until all children of the parallel component are finished. Once they are finished the sub-traces will be scrambled among one and another.

5.2 Experiments

The first part of the experiment setup in this thesis, setting up the experiment data traces, has been completed in the previous section. The second part will detail how the experiments on this data, to test the optimizer and its aspects which are laid out in **Method & Design**, are going to be run. There are three (3) main areas to be tested:

- **MODEL TYPE PERFORMANCE** here the two (2) methods of model construction, as proposed in **Cost Figure Variants**, are going to be compared against one and another on the experiment data. One (1) of the two (2) models, polynomial regression, has several implementation variants which will be compared as well;
- **VARIANT SELECTION AND MERGE METHOD MODES** the nine (9) combinations of variants selection and merge methods, as described in **Variants to optimize selections** and **Merge variant optimizations techniques**, will be compared.
- **OPTIMIZER BEHAVIOUR** Finally, the best performing[2] setup will be used to give an overview of the overall capabilities of the optimizer on all relevant different setups of the small, combined tree datasets. This experiment will also show the resource allocation behaviour depending on its different settings. This final experiment is therefore dependent on the other experiments, and in this thesis it is only setup and described after the analysis of the other experiments has been completed.

For every experiment, all (immediate adjustable) parameters will be recorded. Most experiments will produce larger result datasets which will be interpreted and further distilled down with various statistical methods/operations. The results of these processing steps will be presented in

[2] Note that best performing is based on the metrics described per tested component.

Results. The large intermediary result datasets will be made available in the linked repository (*"https://github.com/MarcelKolen/process-mining-resource-allocation-optimizer"*).

5.2.1 Cost modelling methods Analyses

In this thesis, two (2) methods for cost model discovery have been proposed: linear regression and polynomial regression with variable interaction. The polynomial regression method itself has seven (7) proposed methods to constructs method. Overall, this thesis argues that polynomial regression offers a higher accuracy fit as compared to linear regression, but it is not immediately clear to what degree polynomial regression performs better. And, for the polynomial regression setup, it is not necessarily clear which degree determination method is the best performing method.

This experiment tries to quantify the performance difference both in model accuracy and runtime performance between linear regression and the seven (7) implementation, as described in **Multiple Model Issue**, of the polynomial regression method. Each experiment will be repeated ten (10) times.

5.2.1.1 Comparison metrics

Two (2) elements are going to be explored and tested in the experiments of this section:

- **MODEL TO DATA ACCURACY PERFORMANCE** This will be obtained by using the R^2 and the $RMSE$ methods (***Equation 17*** & ***Equation 18*** respectively). Note that for every run over a dataset, a mean R^2 and $RMSE$ score for that particular dataset is given;
- Time performance For every run of either the linear- or the polynomial regression models over a dataset, the mean average time it took to create the models is recorded.

5.2.1.2 Base settings of components

The **LINEAR REGRESSION MODEL** has three (3) settings which can be altered. These settings are displayed in the table below:

Table 30 – Settings and parameters for the linear regression model method of this experiment.

Exhaustive fit on best model	Split percentage	Split seed
Yes	n.a.	n.a.

The **POLYNOMIAL REGRESSION MODEL** has six (6) settings which can be altered. These settings are recorded in the table below:

Table 31 – Settings and parameters for the polynomial regression model method of this experiment.

Exhaustive fit on best model	Split percentage		Split seed	Degree bound	Best model mode	n-average
Yes	**Train**	$\frac{1}{3}$	n.a.	D_d: [1,5]	inverse R^2 $RMSE$ product	3
	Test	$\frac{2}{3}$				

With the *per model best degree with exhaustive search method* a range of degrees to work within have to be defined. This will be set between one (1) for the lowest degree and five (5) for the highest degree. This figure range has been chosen fairly arbitrarily, however after several test runs on similar (in setup) datasets, as compared to the ones being used for this experiment, whilst testing the polynomial regression method, the highest found degree has not been higher than five (5).

One (1) other setting related to the degree application mode, is the comparison method for determining which degree is better, called *best model mode*. There are three (3) options for this setting, of which the third option has been chosen. These options are described below:

- Compare on $RMSE$, models with lower $RMSE$ values are better;
- Compare on R^2, models with higher R^2 values are better;

- Compare on inverse R^2 $RMSE$ product, this combines $RMSE$ and R^2 via the equation shown in ***Equation 31***. Models with a lower combined score, are better.

$$\overline{R2} \cdot RMSE = (1 - R2) \cdot RMSE$$

Equation 31 – inverse R^2 RMSE product where R^2 is given in **Equation 17** *and RMSE is given in* **Equation 18**.

5.2.2 Variant selection and merge methods comparison

Because this optimizer optimizes on a per variant basis, and then merges the results of the variants into a generalizable solution, methods for selecting the appropriate variants, as described in **Variants to optimize selections**, and methods for merging these results, as described in **Merge variant optimizations techniques**, had to be proposed. In this third experiment the comparison of variant selection strategies and merge methods will take place.

The three (3) variant selection methods to be tested are:

- **ALL VARIANTS**;
- **REQUIRED VARIANTS**;
- **MINIMAL REQUIRED VARIANTS**.

The three (3) merge methods to be tested are:

- **HIGHEST COUNT MERGING**;
- **VARIANT OCCURRENCE WEIGHTED AVERAGE MERGING**;
- **PARETO 80/20 CURVE WEIGHTED AVERAGE MERGING**.

The performance of variant selection cannot be measured without also running one (1) of the merging methods and vice-versa. In order to measure the performances, both improvement impact and time wise, the experiments will be run on all combinations of variants and merge methods. This would result in nine (9) experiments to cover all combinations. Note that this will be done across four (4) different objective modes, of which the results will be merged in an overall dataset:

- **TIME OBJECTIVE**;
- **COST OBJECTIVE**;
- **MULTI-OBJECTIVE TIME CONSTRAINED** this is cost objective optimization with the time objective added as a constraint with a certain epsilon value;
- **MULTI-OBJECTIVE COST CONSTRAINED** this is time objective optimization with the cost objective added as a constraint with a certain epsilon value.

Because of runtime and hardware limitations, this particular experiment will not be repeated ten (10) times like the other experiments. This experiment will only be performed three (3) times.

5.2.2.1 Comparison metrics

There are two (2) metrics worth looking at, the improvement an optimization offers compared against the mean performance of a dataset, and the run time required to achieve said result.

In order to record the run time of the optimizations, first it should be determined what runtime portion of the optimization step should be recorded. As described in **Optimization routine**, there are several steps associated with optimizing a process within the method proposed in this thesis. In this experiment, the only steps of real concern are the variant selection step and the result merging step. The process-, variant- and model discovery part are not taken into account.

The other metric, improvement of an optimized process compared against the mean performance of a dataset, has several aspects worth discussing. First, the mean of a dataset. The mean of a dataset is simply obtained by running an analysis over every dataset and collecting for every case the cost of all

activities and the difference between the start and end time of the first and last activity. These figures are then divided by the number of cases in a dataset to obtain the mean average of a dataset.

The averages of the processes after the optimization results have to be calculated before they can be compared. The optimizer produces a most optimal set of activity-resource combinations and the optimizer also knows, as described in **Variants to optimize selections**, which variants occur for every process, and also how often these variants occur. So, in order to calculate what the mean figures are of the optimized processes, every variant receives a simulated run where the activity-resource combinations suggested by the optimizer are applied. Then, for every variant, the count of the variant is used to repeat the results.

With the baseline performance for every dataset, and with the results for every optimized dataset known, the improvement can be calculated. The improvement is recorded as a percentile improvement. This percentile improvement is measured as follows:

$$\Delta_{\%}(N, O) = \frac{N - O}{|O|}$$

Equation 32 – Percentile difference formula where N signifies a new value and O an old value over which the percentile difference is to be calculated.

In this case, a negative percentile improvement, indicates an improvement by the process optimizer compared to the average of a dataset.

To compare methods on a relative performance level, so across all datasets, a mean average improvement percentile is to be determined. However, as some datasets could produce comparatively large improvement percentiles as compared to other datasets, the results must be normalized. This normalization will be done over the percentile improvements of each tree type with min-max-scaling, as explained in ***Equation 21***, where the feature scale is set to scale between 0.0 and 1.0. Note that the min-max-scaling in this experiment is *not* reversed, unlike in ***Equation 21***.

5.2.2.2 Base settings of components

In order to run this experiment, a cost modelling method needs to be used. This experiment will use the polynomial regression cost modelling method. The settings for the modelling component are to be as follows:

Table 32 – Settings and parameters for the polynomial regression model method of this experiment.

Exhaustive fit on best model	Split percentage		Split seed	Degree determination mode	Degree bound	Best model mode	n-average
Yes	**Train**	$\frac{1}{3}$	n.a.	Per model best degree with exhaustive search	D_d: [1,5]	inverse R^2 $RMSE$ product	n.a.
	Test	$\frac{2}{3}$					

Another setting which needs to be set is the *compromise allowance* for the multi-objective optimization modes. The compromise allowance, which is explained in **Implementation of the ε-constraint method**, will be set to 35% for this experiment.

5.2.3 Optimization improvement & behaviour

This last experiment is a "forward declared" experiment in the sense that this experiment is only formulated after the results from the two (2) previous experiments have been gathered. In this experiment a baseline comparison will be performed to see how well it is able to optimize one (1) of the available datasets. To measure the improvement, the results from the optimizations are to be compared against the average baseline of the given process. This experiment should also show the resource allocation behaviour depending on the used settings for the optimizer and the used dataset. Because this

experiment is going to be used to show the practicality of its results, it will only perform multi-objective optimizations.

The reason this experiment is only formulated after the previous two (2) experiments have been finished and fully analysed, is because in this experiment the best possible setup is to be used.

This experiment will only use a subset of the experiment data. It will use the *Small combined tree setup* (see **Small, combined tree**), using both the *average resources only* (see **Base settings: average resources without random throughput time modifiers**) setup and the *specialized resources* (see **Base settings: throughput time specialized resources** and **Base settings: cost specialized resources**). Datasets with random events have been omitted. They have been omitted because the random events have been added primarily to stress the modelling components. For both data setups, each of the three (3) tree and resource variants will be used. This results in this experiment being run across six (6) different experiment datasets. As said above, the optimizer is to be used in the multi-objective optimization mode only. It will perform the optimizations on two (2) different constraining methods for each dataset, once in throughput time objective constraining, and once for cost objective constraining. The optimizer will use three (3) compromise allowances for each mode. Each experiment configuration will be repeated ten (10) times. This should results in six (6) times two (2) times three (3) times time (10), making 360 results.

5.2.3.1 Comparison metrics

In this experiment there are two (2) elements which will be analysed. One (1) element is the allocation behaviour. This is not so much an element subject to comparisons, but rather it is an analysis into the behaviour of the optimizer. It is expected that when the optimizer is tasked to, for example, optimize more in favour of throughput time performance, that it will select primarily resources which are throughput time efficient.

The other element worth analysing is the improvement over baseline. In this case, improvement should seen as the reduction in the objective values as compared to the average baseline. The improvement will be recorded as a percentual difference using ***Equation 32***. The definition of the average baseline is identical to the definition used in **Comparison metrics**.

5.2.3.2 Base settings of components

MODELLING METHOD The modelling method with the highest accuracy according to the results from **Cost modelling methods Analyses** is the *polynomial modelling* with the following setup:

Table 33 – Settings and parameters for the polynomial regression model method using the single degree method.

Exhaustive fit on best model	Split percentage		Split seed	Degree determination mode	Degree bound	Best model mode	n-average
Yes	**Train**	$\frac{1}{3}$	n.a.	Per model best degree with exhaustive search	D_d: [1,5]	inverse R^2 $RMSE$ product	n.a.
	Test	$\frac{2}{3}$					

VARIANT SELECTION METHOD The best variant selection methods to use according to the results from **Variant selection method performance comparison** are either the *All variants* method or the *Required variants* method. In order to reduce the required runtime, the *Required variants* method will be used as it has a shorted runtime compared to *All variants*.

OPTIMIZATION RESULT MERGING METHOD The most suitable merging method is the *Highest count merging* method as seen in **Optimization results merge methods performance comparison**.

COMPROMISE ALLOWANCE For both multi-objective modes three (3) compromising allowance values are to be used: 30% (0,3), 50% (0,5) and 70% (0,7).

5.3 Runtime environment

This section provides details about the experiment and runtime environment within which all data construction and experimentation tasks are performed. This will be split up in three (3) parts. The first part will be dedicated to describing the runtime hardware. The second part is to be dedicated to the runtime virtualization/platform translation required to run the experiments on the aforementioned hardware. Finally, all versions of the used software and libraries will be formally noted.

5.3.1 Hardware

The hardware on which all data generating, experiments, and analyses are to be performed is described in the table below:

Table 34 – Hardware overview.

Vendor	Model Name	Model ID	Year	CPU	GPU	Memory	Storage
Apple	Macbook Pro (16-inch 2021)	MacBookPro18,2	2021	M1-Max P8E2	M1-Max 32C	64GB Unified Memory	2TB

During all operations, the machine will be set to the "high power" energy mode. The only applications which will be permitted to run are JetBrains-DataSpell and the underlying JuPyter engine, which are used to run and control the operations.

5.3.2 Runtime environment

In order to get the optimizer to work with IBM (Do)CPLEX, a special runtime environment has to be setup. At the current time of performing the experiments, the CPLEX optimizer is only available for the x86-64 versions of MacOS. The used hardware does not use an x86-64 processor, but rather an ARM-64 processor, and could therefore not natively run the CPLEX optimizer.

In order to combat this problem, a binary translation layer will be used. This translation layer, Rosetta 2, is native to the used operating system installed on the hardware. The translation layer is able to convert x86-64 applications, such as IBM's CPLEX optimizer, into ARM-64 compatible software. In order to use the Python modelling tool with CPLEX in x86-64 mode, Python3, and all the associated libraries, will also run as x86-64 applications through the Rosetta 2 conversion layer.

Finally, on the topic of the CPLEX optimizer, all optimization runs are done using the *constraint programming* engine from CPLEX. Both the linear and the polynomial optimization experiments are performed using this engine. This is done because the constraint programming engine offers more flexibility and supports both linear as well as polynomial cost model.

5.3.3 Software and library versions

This section includes the versions of all relevant software packages/systems and libraries which will be used to run all operations on. Note that dependency libraries are not included, only primary libraries.

Table 35 – Software and libraries overview.

Name		Version
MacOS Monterey		12.5.1 (21G83)
CPlexStudio		2210
Dataspel 2021.3.3		#DS-213.7172.29
Jupyter		1.0.0
Python		3.10.4
[3]	**CPlex**	22.1.0.0

[3] These following entries are libraries used within Python.

	DoCplex	2.23.221
	Numpy	1.23.1
	Pandas	1.4.3
	Pm4py	2.2.24
	Scikit-learn	1.1.1
	Scipy	1.8.1

6 Results

In this section all results and analyses from the experiments, described in **Experiments** which were run on the data generated in **Experiment Data**, are going to be presented. The results will be accompanied with a brief interpretation and conclusive discussion. This discussion should provide the reader with meaningful insights in the collected results.

6.1 Cost modelling methods Analyses

The experiment run, described in **Cost modelling methods Analyses**, resulted in 4,800 by 12 intermediary results. These results will be analysed and compiled in a more digestible fashion. For this experiment two (2) distinct analyses will be performed.

MEAN AVERAGE AND STANDARD DEVIATION MODEL ACCURACY COMPARISON The first analysis is aimed at quantifying the accuracy performance differences between the eight (8) different modelling techniques. This will be done by compiling the mean average R^2 and $RMSE$ scores for the eight (8) models. Besides the mean average, the standard deviation for both metrics will be given as well.

This will be done across three (3) dataset splits:

- **OVERALL DATASET PERFORMANCE**;
- **Performance exclusively on datasets with random events**;
- **Performance exclusively on datasets without random events**.

The latter two (2) will only be compared on the throughput time objective, as this is the only part of the modelling task which has random events added to its datasets. Refer to **Resource simulation values** for the complete setup of the random event inclusion.

MODELLING TIME MEAN AVERAGE the second analysis is aimed at comparing the required run time to construct the linear and polynomial models. This analysis compares the mean average of the eight (8) models. In giving the mean average, the standard deviation will also be given.

The result tables below use abbreviation notations for the model descriptors. The following table explains the different abbreviations.

Table 36 – Abbreviation list for the Polynomial Regression methods.

SD	BAD	BADg	BnAD	BnADg	PMBDe	PMBDg
Single Degree	Best Average Degree	Best Average Degree Greedy	Best n Average Degree	Best n Average Degree Greedy	Per Model Best Degree Exhaustive	Per Model Best Degree Greedy

6.1.1 R2 & RMSE mean average and standard deviation model accuracy comparison

In comparing the models on accuracy there are two (2) accuracy metrics (R^2 and $RMSE$) to be considered. The results will be compared for both metrics separately. In each comparison, three (3) methods are to be compared: the linear method, the worst performing polynomial method and the best performing polynomial method. The comparison will be split between the cost and throughput time objective where-ever applicable.

After comparing the models, a short conclusive discussion will be held. This discussion should provide a definitive answer on which method is the best for accuracy and the compromises are.

6.1.1.1 Overall data

Table 37 – Model accuracy comparison analyses between linear and polynomial regression models, using the R^2 and RMSE metrics. n signifies the number of datapoints these comparisons are gathered from. Please refer to **Table 36** *for a list of abbreviation definitions used for the models.*

Model	*Objective mode*	*n*	Mean R^2	Std.[4] R^2	Min R^2	Max R^2	Mean *RMSE*	Std. *RMSE*	Min *RMSE*	Max *RMSE*
Lin[5]	**Throughput Time Objective**	300	0.892	0.138	0.486	0.999	25.505	38.882	$4.366 \cdot 10^{-14}$	177.944
SD		300	0.915	0.146	0.486	1.000	4.919	5.654	$4.511 \cdot 10^{-14}$	18.882
BAD		300	0.911	0.145	0.486	1.000	8.070	7.320	$4.660 \cdot 10^{-14}$	40.454
BADg		300	0.910	0.144	0.486	1.000	9.641	9.353	$4.542 \cdot 10^{-14}$	69.433
BnAD		300	0.913	0.145	0.486	1.000	7.184	7.722	$4.584 \cdot 10^{-14}$	61.055
BnADg		300	0.914	0.146	0.486	1.000	6.308	6.340	$4.546 \cdot 10^{-14}$	28.710
PMBDe		300	0.914	0.146	0.486	1.000	6.358	6.621	$4.558 \cdot 10^{-14}$	40.201
PMBDg		300	0.912	0.145	0.486	1.000	8.101	9.536	$4.518 \cdot 10^{-14}$	71.133
Lin	**Cost Objective**	300	0.935	0.106	0.604	0.999	1.390	1.564	$2.710 \cdot 10^{-14}$	6.601
SD		300	0.939	0.108	0.605	1.000	0.005	0.018	$2.699 \cdot 10^{-14}$	0.084
BAD		300	0.938	0.108	0.604	1.000	0.276	0.361	$2.775 \cdot 10^{-14}$	2.460
BADg		300	0.938	0.108	0.604	1.000	0.350	0.429	$2.775 \cdot 10^{-14}$	2.598
BnAD		300	0.939	0.108	0.605	1.000	0.135	0.213	$2.766 \cdot 10^{-14}$	1.350
BnADg		300	0.939	0.108	0.604	1.000	0.111	0.172	$2.775 \cdot 10^{-14}$	1.080
PMBDe		300	0.939	0.108	0.605	1.000	0.117	0.196	$2.791 \cdot 10^{-14}$	1.111
PMBDg		300	0.939	0.108	0.605	1.000	0.005	0.018	$2.699 \cdot 10^{-14}$	0.084

Comparing the methods above, the *linear regression* method, the *Polynomial: Best Average Degree Greedy* and the *Polynomial: Single Degree* methods will be considered.

Table 38 – Three-way model comparison conclusion on the R^2 and RMSE metrics.

Best and worst methods on mean average R^2 performance			
Objective	**Linear regression**	**Polynomial: BADg**	**Polynomial: SD**
Time	0.892	0.910 (+2.02 %)	0.915 (+2.58 %)
Cost	0.935	0.938 (+0.32 %)	0.929 (+0.43 %)
Best and worst methods on mean average *RMSE* performance			
Objective	**Linear regression**	**Polynomial: BADg**	**Polynomial: SD**
Time	25.505	9.641 (-62.20 %)	4.919 (-80.71 %)
Cost	1.390	0.350 (-74.82 %)	0.005 (-99.64 %)

COMPARING THE LINEAR REGRESSION MODELS AGAINST THE POLYNOMIAL MODELS on the R^2 metric, the polynomial models score better on both objectives, but only by a small margin. The time

[4] "Std." is used as an abbreviation for *Standard Deviation*.
[5] "Lin" is used as a shorthand for *Linear Regression*.

objective sees the largest difference, and this is most likely due to the more complex nature of the time objective setup and experiment data. The *RMSE* metric sees a larger deviation of the accuracy performance between the three (3) modelling methods, especially between the linear and the polynomial models. On both metrics, the worst performing polynomial model is the *Best Average Degree Greedy* model, and the best model is the *Single Degree* model. Overall, the polynomial models perform better than the linear models, but depending on the metric, only by a small margin.

A NOTABLE ABSENCE from the comparisons is the *Per Model Best Degree Method Exhaustive* method. Earlier in this thesis an expectancy was given that this method will perform the best of all methods. The primary idea behind this prediction was that this method should have been able to prevent overfitting and thereby beat a method such as the *Single Degree* method which is more susceptible to overfitting. It should be noted that this method is still among the most accurate methods, but nevertheless, there are two (2) most probable explanations behind these results:

- **THE FIRST REASON** is that at the current degree depth (max of five (5)) on the used datasets, overfitting is not as big as an issue yet, so, the model which always applies the most powerful model (*Single Degree* in this case) will theoretically be the best performer. If the degree budget was increased and/or with a different dataset, overfitting might have become an issue where the *Per Model Best Degree Method Exhaustive* method would have been able to pull ahead;
- **THE SECOND REASON** why the *Per Model Best Degree Method Exhaustive* method might underperform compared to the *Single Degree* method, is because it bases its degrees on a development on a data-subset for every activity-resource model. By chance, this subset might have been selected unfavourably which could results in an ill-fitting degree for the overall model.

6.1.1.2 Exclusively with vs without random events

Table 39 – Model accuracy comparison analyses between linear and polynomial regression models, using the R^2 and RMSE metrics. n signifies the number of datapoints these comparisons are gathered from. Please refer to **Table 36** *for a list of abbreviation definitions used for the models. These results are based only on the throughput time objective. The "Dataset" column indicates whether the random or non-random dataset is used.*

Model	*Dataset*	*n*	Mean R^2	Std. R^2	Min R^2	Max R^2	Mean *RMSE*	Std. *RMSE*	Min *RMSE*	Max *RMSE*
Lin	**With Random Events**	300	0.867	0.161	0.486	0.983	28.990	40.059	3.576	177.944
SD		300	0.890	0.171	0.486	0.996	9.640	4.337	3.338	18.882
BAD		300	0.887	0.170	0.486	0.996	11.582	6.044	3.517	31.108
BADg		300	0.886	0.169	0.486	0.996	12.771	8.409	3.576	69.433
BnAD		300	0.888	0.171	0.486	0.996	11.452	7.190	3.350	46.973
BnADg		300	0.889	0.171	0.486	0.996	10.625	5.495	3.345	28.710
PMBDe		300	0.889	0.171	0.486	0.995	10.609	5.450	3.345	31.022
PMBDg		300	0.887	0.170	0.486	0.995	12.259	9.989	3.344	71.133
Lin	**Without Random Events**	300	0.917	0.103	0.605	0.999	22.021	34.104	$6.048 \cdot 10^{-14}$	164.990
SD		300	0.940	0.110	0.605	1.000	0.198	5.654	$4.511 \cdot 10^{-14}$	3.566
BAD		300	0.936	0.109	0.605	1.000	4.558	7.320	$4.660 \cdot 10^{-14}$	40.454
BADg		300	0.934	0.108	0.605	1.000	6.511	9.353	$4.542 \cdot 10^{-14}$	42.865
BnAD		300	0.939	0.110	0.605	1.000	2.917	7.722	$4.584 \cdot 10^{-14}$	61.055
BnADg		300	0.939	0.110	0.605	1.000	1.990	6.340	$4.546 \cdot 10^{-14}$	24.399
PMBDe		300	0.939	0.110	0.605	1.000	2.108	6.621	$4.558 \cdot 10^{-14}$	40.201

PMBDg		300	0.938	0.109	0.605	1.000	3.944	9.536	$4.518 \cdot 10^{-14}$	33.583

In the random and non-random dataset comparisons, three (3) modelling techniques will be compared. The three (3) models which will be compared are: the *linear regression* method, the *Polynomial: Best Average Degree Greedy* and the *Polynomial: Single Degree* methods.

Table 40 – Three-way model comparison conclusion on the R^2 and RMSE metrics.

Best and worst methods on mean average R^2 performance			
Dataset	**Linear regression**	**Polynomial: BADg**	**Polynomial: SD**
Random	0.867	0.886 (+2.19 %)	0.890 (+2.65 %)
Non-Random	0.917	0.934 (+1.85 %)	0.940 (+2.51 %)
Best and worst methods on mean average *RMSE* performance			
Objective	**Linear regression**	**Polynomial: BADg**	**Polynomial: SD**
Random	28.990	12.771 (-55.95 %)	9.640 (-66.75 %)
Non-Random	22.021	6.511 (-70.43 %)	0.198 (-99.10 %)

For both datasets, the polynomial methods again deliver a higher accuracy performance as compared to the linear method. For all methods goes that adding random elements to the datasets hurts the accuracy performance of the models.

The two (2) accuracy metrics do not pain a clear picture whether one (1) model is better at dealing with random data than the other. In the R^2 metric comparisons, the polynomial methods show a higher percentual improvement on data with random elements as compared to data without random elements. But, for the $RMSE$ metric the opposite is true. This shouldn't be surprising as regression is not meant to work with and model for random outliers, it should model patterns. Instead of using a more complex modelling technique, such as polynomial regression, random noise in data should probably be pre-processed out of the data in order to improve model accuracy on noisy data.

6.1.2 Modelling time mean average and standard deviation comparison

This overview provides the mean average runtime of each of the eight (8) models and the percentual difference between, using the linear model as the baseline as that is the fastest model.

Table 41 – Modelling run time comparison analyses between linear and polynomial regression across both objectives (throughput time and cost), where n signifies the number of datapoints these comparisons are gathered from. Please refer to **Table 36** *for a list of abbreviation definitions used for the models.*

Model	***n***	**Mean (s)**	**Standard Deviation**	**Min (s)**	**Max (s)**
Linear	600	10.907 (+-0.00%)	6.228	2.454	28.818
SD	600	12.884 (+18.13%)	0.110	2.859	39.355
BAD	600	12.507 (+14.67%)	0.109	2.770	31.790
BADg	600	12.280 (+12.59%)	0.108	2.714	31.147
BnAD	600	12.736 (+16.77%)	0.110	2.770	31.143
BnADg	600	13.130 (+18.13%)	0.110	2.812	31.343
PMBDe	600	13.491 (+20.38%)	0.110	3.091	36.223
PMBDg	600	13.029 (+19.46%)	0.109	2.773	35.060

As stated above, the *linear* method is the fastest method overall, and of the polynomial methods the *Best Average Degree Greedy* method is the fastest whereas the *Per Model Best Degree Method Exhaustive* is the slowest overall. These results are all fairly in line of expectations, especially if the complexity of the methods is to be taken into regard.

The *linear* method is the simplest method which creates simple models without performing any hyper parameter tuning. The *Best Average Degree Greedy* method develops a best overall degree for the entire model set in a greedy manner, so every remaining model outside of the development set has only one (1) degree to develop for without any further hyper parameter tuning. But the same is true for the *Single Degree* approach, which also only applies one (1) degree for all models, so then why is *Best Average Degree Greedy* faster by roughly 6 percent points? The most probable explanation is that the *Best*

Average Degree Greedy method manages to develop a degree which is smaller than the degree used in the current *Single Degree* setup, and atop of that, that a lower degree has a higher impact on runtime than having to develop a small model subset greedily.

The *Per Model Best Degree Method Exhaustive* method being the slowest isn't surprising either, because this method develops, in the current setup with five (5) degree depths, five (5) different models for each activity-resource model and then evaluates the best performing degree depth. This method develops the most models, including the most complex models (with the highest degree depth), so it is no surprise that this method is slower.

6.1.3 Results discussion

There are three (3) main conclusions/findings which can be drawn from this experiment:

- **LINEAR VS. POLYNOMIAL** The results show that, overall, the polynomial models perform better than the linear models. The accuracy improvement is greater on the $RMSE$ metric than on the R^2 metric, however, still in both models it shows an overall improvement in favour of the polynomial models. Of these models the *Single Degree* method seems to be the most effective for the current setup. Note that in a different setup (different data or a greater degree depth budget) the *Per Model Best Degree Method Exhaustive* method could take over as the better method. As a last note, yes, the polynomial models perform better, but the linear method are also a surprisingly strong candidate. This means that it is still feasible to use this approach as a fully linear optimizer, which could constitute it as a NP-Complete problem as compared to an NP-Hard problem as stated in **Problem Type**. A potential explanation for the small difference between the linear and polynomial model performance could be down to the used data. The data models exhibit fairly linear behaviour, such as the long end of a root function, or the threshold functions described in **Activity simulation values** and **Resource simulation values**. If more complex models were used with fewer linear components, then an increase in performance difference might have been more noticeable;
- **RANDOM VS. NON-RANDOM** In the results from the dataset split on datasets with random and without random elements added to them, the conclusion could be drawn that applying polynomial models is not a solution to dealing with random noise in the data. Whilst the polynomial models did perform better than the linear models in random data, they also do that on non-random data, so it is not possible to draw a conclusion there to say that polynomial models solve randomness in data. The better solution would probably be to pre-process the data to eliminate random noise before applying the cost models;
- **TIME PERFORMANCE** Comparing the time performance is simple, linear is the fastest method, and the polynomial methods, and especially *Per Model Best Degree Method Exhaustive*, are slower overall. More important perhaps is to wonder what a longer runtime offers. Whilst the *Single Degree* method was 18.13% slower than the linear method, it does offer higher accuracy, up to 2.58 % on the R^2 metric (overall data time-objective) and up to 99.64 % the $RMSE$ metric (overall data cost-objective). This makes the runtime difference comparatively small. Arguably, the linear method should only be used if the model is to remain exclusively linear, else polynomial methods should be used.

6.2 Variant selection and merge methods results

The experiment run, described in **Variant selection and merge methods comparison**, resulted in 6.353 by 16 intermediary results. These results will be analysed and compiled in a more digestible fashion. For this experiment three (3) distinct analyses will be performed.

- **COMBINATIONS OF OPTIMIZATION RESULT MERGE METHODS AND VARIANT SELECTION METHODS** This analysis attempt to compare all nine (9) combinations of the merge and variant selection methods against one and another.

- **VARIANT SELECTION METHODS ONLY** This analysis attempt to compare the three (3) different variant selection methods against one and another. This comparison is expected to show the largest variance in runtime as compared to the variances between different merge methods, meaning this is most likely the setting with the largest impact on runtime performance.
- **OPTIMIZATION RESULT MERGE METHODS ONLY** This analysis attempt to compare the three (3) different optimization result merge methods against one and another. This comparison is expected to show the largest variance in objective function improvement as compared to the variant selection method, meaning this is most likely the setting with the largest impact on optimization improvement performance, up to

The experiments are run across four (4) objective modes as is mentioned in the introduction of this experiment. The mean average process improvement performances and the mean average run time performances of these four (4) objectives is pooled together in overall results in order to generalize the outcomes.

Every result is sorted from best to worst. Note that in all cases, lower values (such as min-max-scaled percentile objective improvements) are better than higher values. This is because all results are feature scaled between zero (0) and one (1), which means that negative percentage differences (which indicate an improvement in the objectives) are closer to zero (0) as compared to positive percentage difference (which indicate a worsening of the objectives).

The results are discussed per section. Merge methods and variant selection finds their results in their own dedicated subsections.

6.2.1 Variant and merge combined performance comparison

OBJECTIVE FUNCTION IMPROVEMENT PERFORMANCE

*Table 42 – Overall performance comparison of variant selection and merging method combinations. The results are sorted best to worst by means of performing a two (2) field sort on **time mean difference mean** and **cost mean difference mean**.*

#	Variant selection	Merging methods	Time mean difference mean	Cost mean difference mean	Run time mean
0	**Required variants**	**Highest count merge**	0.498081	0.344647	0.149909
1	**Required variants**	**Weighted average merge**	0.500621	0.348941	0.144596
2	**All variants**	**Highest count merge**	0.502408	0.334211	0.373876
3	**Required variants**	**Pareto merge**	0.504904	0.349151	0.14591
4	**All variants**	**Pareto merge**	0.505721	0.346207	0.363251
5	**Min required variants**	**Weighted average merge**	0.509784	0.346696	0.142106
6	**All variants**	**Weighted average merge**	0.51053	0.348687	0.383396
7	**Min required variants**	**Pareto merge**	0.51393	0.345434	0.141287
8	**Min required variants**	**Highest count merge**	0.515488	0.348155	0.130417
			Standard Deviation		
			0.005993	0.004633	0.115796

The result overview in the table above shows which combinations of *variants to optimize selection methods* and *optimization results merge methods* perform best overall in terms of mean differences to the cost and throughput time objective baselines.

On average and overall, the best combination to use is the *required variants* variant selections method and the *highest count* merging method. In the overall setup, the *highest count* merge method was not necessarily the method which was expected to perform best, but, as can be seen in the merge method comparison below, this result is consistent across both analyses. The variant selection method does not necessarily perform the best compared to the second analysis. However, on both fronts, something which should be noted is that the deviation between performance results is fairly small between all combinations. It is also not immediately clear which component *variants to optimize selection methods* or *optimization results merge methods* has the highest overall impact on process improvement.

RUNTIME PERFORMANCE

The runtime performance is determined by sorting the run time mean column in the table above from lowest to highest value. The index order, thereby the performance order, is as follows:

$$8, 7, 5, 1, 3, 0, 4, 2, 6$$

The runtime performance is most likely influenced greatest by the variant selection method, as that influences the subset size runtime complexity. It is therefore not surprising to see that the runtime performance sorting is grouped precisely against the three (3) variant selection methods. The fastest three (3) results $(8, 7, 5)$ are given by the *min required variants* method, the slowest three (3) results $(4, 2, 6)$ are given by the *all variants* method, and the remaining three (3) middle performers $(1, 3, 0)$ are all the *required variants* method. These results are also consistent with the findings below.

6.2.2 Variant selection method performance comparison

OBJECTIVE FUNCTION IMPROVEMENT PERFORMANCE

*Table 43 – Overall performance comparison of variant selection methods. The results are sorted best to worst by means of performing a two (2) field sort on **time mean difference mean** and **cost mean difference mean**.*

#	Variant selection	Time mean difference mean	Cost mean difference mean	Run time mean
0	**All variants**	0.50622	0.343035	0,373508
1	**Min required variants**	0.513067	0.346762	0,137937
2	**Required variants**	0.501202	0.34758	0,146805
		Standard Deviation		
		0.005956	0.002422	0.133521

Among the three (3) selection methods *all variants* is the best performing method in terms of objective improvement, but, it should be noted that the difference between the three (3) methods is minute. Because the differences are small, and because the results are conflicting with earlier results, no real general recommendation could be given on which method is provides the best overall objective improvement. At most: all variants and required variants should provide more data to create a more representative outcome.

RUNTIME PERFORMANCE

The runtime performance is determined by sorting the run time mean column in the table above from lowest to highest value. The index order, thereby the performance order, is as follows:

$$1, 2, 0$$

The runtime performance is consistent with earlier results, where the *min required variants* method is the fastest and the *all variants* method is the slowest. This is no surprise as the *min required variants* method has the lowest number of optimizations to be run, then the *required variants* method and lastly the *all variants* method. If a full data coverage is to be desired, then the *all variants* method should be used, otherwise, because the difference between the lather two (2) methods is small (~1%), the *required*

variants method should preferably be used as it allows for a more larger, and potentially more representative, sample. Only if runtime is critical should the *min required variants* method be used.

6.2.3 Optimization results merge methods performance comparison

OBJECTIVE FUNCTION IMPROVEMENT PERFORMANCE

*Table 44 – Overall performance comparison of optimization result merge methods. The results are sorted best to worst by means of performing a two (2) field sort on **time mean difference mean** and **cost mean difference mean**.*

#	Merging methods	Time mean difference mean	Cost mean difference mean	Run time mean
0	**Highest count merge**	0.505326	0.342338	0,218067
1	**Pareto merge**	0.508185	0.346931	0,216816
2	**Weighted average merge**	0.506978	0.348108	0,223366
		Standard Deviation		
		0.001435	0.003049	0.003477

For the merging methods the *highest count merge* method is overall best performer for both objective functions. This is consistent with earlier results, but not necessarily as predicted earlier in this thesis. This is a non-weighted method, so it does not necessarily optimize for more prevalent variants. Instead, it's an equal voting system. The most likely explanation for this higher performance as compared to weighted methods is that this method takes less prevalent variants equally in regards as more often occurring variants. This approach might improve the performances in worst case process instances, which could (significantly) improve the average performance. But what should be noted overall, is that these methods are all fairly close together in terms of performance. This could indicate these methods are not optimal and that better methods, which for example have a fairer weight application, could exist.

RUNTIME PERFORMANCE

The runtime performance is determined by sorting the run time mean column in the table above from lowest to highest value. The index order, thereby the performance order, is as follows:

$$1, 0, 2$$

The runtime performance is only partially consistent with earlier results. The difference in runtime performance is fairly minute, and based on their theoretical implementation, these methods shouldn't create large runtime differences between the different methods. Because of that, no recommendation on runtime performance can be given for the merging methods.

6.3 Optimization improvement & behaviour results

From the experiment described in **Optimization improvement & behaviour** there are two (2) results to be analysed. First the improvement over baseline. As there are 360 results, there are also 360 baseline differences. These results will be reduced in to two (2) by six (6) categories making for 12 results.

The first distinction is results over the datasets with only average performing resources or over the datasets with average performing resources and specialized resources. The second distinction is made over the two (2) optimization modes, where in the first mode the optimizer compromises on the throughput time and in the second mode the optimizer compromises on the cost. For each of these modes, there are three (3) set compromise allowances, and the results will be presented against these compromise allowances.

The results which will be presented are the mean averages and standard deviations over the different tree variants and repeated runs. The mean averages and standard deviations will be given for the improvement percentage over the throughput time and the cost objectives.

The second results to analyse is the allocation behaviour. It's expected that allocation behaviour changes based on three (3) factors: The used resource pool (only average performers or average performers with specialized resources), the multi-objective compromising mode, and the compromise allowance. There are, just like with improvement over baseline performance, 12 different combinations to be made between the datasets and the different objective modes and compromise allowances. However, there is one (1) extra element which needs to be considered. Each tree, and therefor each resource set, has three (3) different versions. This means that in total there will be 36 results to be presented. The results are presented as a matrix where the activities are presented as columns and the allocated resource is presented in different rows.

As there are ten (10) repeat runs, the way in which the resources are allocated might change between runs. This will be taken into account by showing the allocation occurrence percentage in case that multiple resources are allocated to activities.

6.3.1 Difference against baseline performance

In this section four (4) results are presented in two (2) tables which indicate the mean overall objective differences as compared to the baseline which the optimizer is able to obtain for given configurations. Following these results, a brief results discussion is conducted.

6.3.1.1 Only average resources

Table 45 – Average performance difference achieved by optimizer over baseline.

Tree	Obj[6]	Compromise Allowance	Mean Time Improvement (%)	Time Improvement Standard Deviation	Mean Cost Improvement (%)	Cost Improvement Standard deviation	n
0	Multi-objective: Time Objective Constrained[7]	30 %	16.69 %	$2.93 * 10^{-15}$	-26.48 %	0.00	10
		50 %	16.69 %	$2.93 * 10^{-15}$	-26.60 %	0.00	10
		70 %	16.69 %	$2.93 * 10^{-15}$	-26.49 %	$5.85 * 10^{-15}$	10
1		30 %	-8.14 %	0.00	-18.90 %	$2.93 * 10^{-15}$	10
		50 %	-4.47 %	$7.31 * 10^{-16}$	-19.58 %	0.00	10
		70 %	-4.51 %	0.00	-19.57 %	$2.93 * 10^{-15}$	10
2		30 %	-10.16 %	$1.46 * 10^{-15}$	-25.22 %	0.00	10
		50 %	-9.93 %	0.00	-25.22 %	0.00	10
		70 %	-9.93 %	0.00	-25.22 %	0.00	10
0	Multi-objective: Cost Objective Constrained[8]	30 %	-21.69 %	$2.93 * 10^{-15}$	-19.01 %	$2.93 * 10^{-15}$	10
		50 %	-21.71 %	$2.93 * 10^{-15}$	-16.45 %	0.00	10
		70 %	-21.70 %	$2.93 * 10^{-15}$	-14.72 %	0.00	10
1		30 %	-18.55 %	0.00	-12.71 %	$2.93 * 10^{-15}$	10
		50 %	-17.48 %	0.00	-9.80 %	0.00	10
		70 %	-17.50 %	$2.93 * 10^{-15}$	-11.57 %	$1.46 * 10^{-15}$	10
2		30 %	-21.77 %	0.00	-19.20 %	0.00	10
		50 %	-24.65 %	$2.93 * 10^{-15}$	-14.22 %	0.00	10
		70 %	-26.85 %	0.00	-10.75 %	$1.46 * 10^{-15}$	10

[6] "Obj" is shorthand for *Objective mode*.
[7] Note that the optimizer is optimizing for the *Cost objective* and using the *Throughput time* as a constraint.
[8] Note that the optimizer is optimizing for the *Throughput time objective* and using the *Cost* as a constraint.

6.3.1.2 Average and specialized resources

Table 46 – Average performance difference achieved by optimizer over.

Tree	Obj	Compromise Allowance	Mean Time Improvement (%)	Time Improvement Standard Deviation	Mean Cost Improvement (%)	Cost Improvement Standard deviation	n
0	Multi-objective: Time Objective Constrained	30 %	-21.04 %	0.00	-49.04 %	$5.85 * 10^{-15}$	10
		50 %	3.18 %	0.00	-55.88 %	0.00	10
		70 %	8.72 %	0.00	-56.19 %	0.00	10
1		30 %	-52.70 %	0.00	-11.69 %	0.00	10
		50 %	12.78 %	0.00	-52.59 %	$1.17 * 10^{-14}$	10
		70 %	29.32 %	0.00	-52.84 %	0.00	10
2		30 %	-27.722 %	$5.85 * 10^{-15}$	-42.51 %	$5.85 * 10^{-15}$	10
		50 %	0.28 %	0.00	-58.26 %	$1.17 * 10^{-14}$	10
		70 %	14.08 %	$2.93 * 10^{-15}$	-58.01 %	$1.17 * 10^{-14}$	10
0	Multi-objective: Cost Objective Constrained	30 %	-29.65 %	$5.85 * 10^{-15}$	-48.00 %	$5.85 * 10^{-15}$	10
		50 %	-25.40 %	$5.85 * 10^{-15}$	-7.48 %	0.00	10
		70 %	-47.87 %	$5.85 * 10^{-15}$	-29.56 %	0.00	10
1		30 %	2.43 %	$3.66 * 10^{-16}$	-31.80 %	0.00	10
		50 %	-53.50 %	0.00	-12.38 %	0.00	10
		70 %	-56.08 %	0.00	3.21 %	0.00	10
2		30 %	-35.11 %	0.00	-41.25 %	0.00	10
		50 %	-50.38 %	0.00	-19.90 %	0.00	10
		70 %	-56.94 %	$1.17 * 10^{-14}$	-9.57 %	0.00	10

6.3.1.3 Results discussion

Analysing the results above, a few interesting findings can be extracted. It should first be noted that some of the results using the current dataset and configuration setup does not always provide a clear picture. There are however still other results to extract findings from:

- **IMPROVE OBJECTIVES** The first finding is that the optimizer is able to improve both objectives over the performance given by the baseline. There are a few exceptions, such as for tree version zero (0) for the time objective constrained setup in ***Table 45***, there the optimizer is not able to improve both objectives for any setting;
- **COMPROMISE ALLOWANCE** Looking into the compromise allowance, the key parameters for the multi-objective optimization part of this optimizer, then a strong pattern can be observed. In both results, but more so in the results from the runs with specialized resources (***Table 46***), the effect of changing the compromise allowance parameter can be seen in the balance of improvement between the two (2) objectives. Taking for example tree version two (2) for the cost objective constrained setup in ***Table 46*** (last three (3) rows), this balance slide between the two (2) objectives is strongly noticeable. In these three (3) results the compromise allowance is applied to the cost objective, meaning that as the compromise allowance increases, the optimizer is allowed to "neglect" the cost objective more and focus more on the time objective. At 30% compromising allowance, the cost objective sees a higher percentual improvement than the time objective, but as this allowance is increased, the performance for the cost objective decreases whereas it increases for the time objective. The compromise allowance therefor seems to be effective;
- **DIMINISHING RETURNS** Staying with the compromise allowance, some setups show a limited return on increasing the compromise allowance beyond a certain point. An example is the setup for tree version one (1) for the cost objective constrained setup in ***Table 46***. Increasing the allowance from 30% to 50%, the time objective sees north of a 50 percent point performance improvement. However, from 50% to 70% there is only a +-3 percent point improvement, whilst at the same time the cost objective is compromised so much that the optimizer now worsens the

cost objective compared to the baseline. This limited and diminishing return is most likely a characteristic of the used dataset. When referring back to the example from ***Figure 11*** in **Multi-objective optimization applicability**, a clear levelling out of the Pareto optimum front can be observed where one (1) objective is only slightly improved, but the other objective is strongly compromised;

- **OBJECTIVE MODE PREFERENCE** When observing the objective mode selection, an overall image emerges that the objective which is optimized for, will be better off than the objective which is constrained. This is perhaps not a complete surprise, but it does introduce a weakness of the ε-constraint method.

6.3.2 Allocation behaviour

In this section four (4) results are presented in two (2) tables which indicate the mean overall objective differences as compared to the baseline which the optimizer is able to obtain for given configurations. Following these results, a brief results discussion is conducted.

An important note to make before the analysis and discussion can commence is the following: resources zero (0, *"r_0"*) up to and until resource nine (9, *"r_9"*) are all average performance resources. Some perform better in a certain objective than the other, but overall, they all roughly perform equally for the objectives. Resources 10 (*"r_10"*) up to and until resource 13 (*"r_13"*) are all throughput time specialized resources. Resources 14 (*"r_14"*) up to and until resource 17 (*"r_17"*) are all cost specialized resources. This goes for all configurations in this experiment. The specific resource configurations and setups can be found in **10.13.4**.

6.3.2.1 Only average resources

Table 47 – Resource allocation behaviour of the optimizer. The columns headers starting with 'a_' denote the activities, and the cell elements starting with 'r_' denote the resources. Between runs, the suggested resource for an activity might change. If several resources are suggested between the runs, the resources will be noted as: 'r_0: 75%, r_4: 20%, r_5: 5%' which means that resource zero (0) is used 75% of the time, four (4) only 20% and finally five (5) just 5%. If no percentage is shown, then that resource is used 100%.

			Resources allocated to activities							
Tree	**Obj**	**Compromise Allowance**	**a_a**	**a_b**	**a_c**	**a_d**	**a_e**	**a_f**	**a_g**	**n**
0	**Multi-objective: Time Objective Constrained**	30 %	r_0	r_0	r_0	r_0	r_0	r_3	r_0	10
		50 %	r_0	r_0	r_0	r_0	r_0	r_3	r_0	10
		70 %	r_0	r_0	r_0	r_0	r_0	r_3	r_0	10
1		30 %	r_6	r_1	r_6	r_0	r_0	r_6	r_1	10
		50 %	r_6	r_1	r_6	r_0	r_1	r_6	r_1	10
		70 %	r_6	r_1	r_6	r_0	r_1	r_6	r_1	10
2		30 %	r_6	r_6	r_6	r_6	r_0	r_0	r_0	10
		50 %	r_6	r_6	r_6	r_6	r_0	r_0	r_0	10
		70 %	r_6	r_6	r_6	r_6	r_0	r_0	r_0	10
0	**Multi-objective: Cost Objective Constraine**	30 %	r_0	r_1	r_3	r_9	r_0	r_6	r_0	10
		50 %	r_0	r_1	r_0	r_4	r_9	r_3	r_6	10
		70 %	r_0	r_1	r_6	r_9	r_0	r_6	r_4	10
1		30 %	r_8	r_1	r_4	r_0	r_3	r_8	r_6	10
		50 %	r_9	r_1	r_4	r_1	r_6	r_7	r_5	10
		70 %	r_6	r_1	r_4	r_1	r_6	r_7	r_7	10
2		30 %	r_6	r_5	r_5	r_6	r_0	r_0	r_6	10
		50 %	r_8	r_3	r_1	r_6	r_0	r_0	r_3	10
		70 %	r_8	r_3	r_1	r_7	r_0	r_0	r_6	10

6.3.2.2 Average and specialized resources

MULTI-OBJECTIVE: TIME OBJECTIVE CONSTRAINED

Table 48 – Resource allocation behaviour of the optimizer. The columns headers starting with 'a_' denote the activities, and the cell elements starting with 'r_' denote the resources. Between runs, the suggested resource for an activity might change. If several resources are suggested between the runs, the resources will be noted as: 'r_0: 75%, r_4: 20%, r_5: 5%' which means that resource zero (0) is used 75% of the time, four (4) only 20% and finally five (5) just 5%. If no percentage is shown, then that resource is used 100%.

			Resources allocated to activities							
Tree	**Obj**	**Compromise Allowance**	**a_a**	**a_b**	**a_c**	**a_d**	**a_e**	**a_f**	**a_g**	**n**
0	**Multi-objective: Time Objective Constrained**	30 %	r_16	r_12	r_16	r_17	r_17	r_16	r_16	10
		50 %	r_15	r_16	r_15	r_14	r_14	r_15	r_14	10
		70 %	r_16	r_17	r_14	r_15	r_17	r_16	r_15	10
1		30 %	r_15	r_12	r_13	r_17	r_15	r_15	r_16	10
		50 %	r_15	r_14	r_15	r_16	r_16	r_16	r_16	10
		70 %	r_17	r_17	r_14	r_15	r_17	r_14	r_16	10
2		30 %	r_15	r_15	r_17	r_15	r_13	r_0	r_15	10
		50 %	r_16	r_16	r_17	r_16	r_17	r_15	r_15	10
		70 %	r_17	r_14	r_17	r_14	r_15	r_17	r_14	10
0	**Multi-objective: Cost Objective Constraine**	30 %	r_14	r_11	r_16	r_3	r_17	r_16	r_14	10
		50 %	r_6	r_13	r_17	r_12	r_12	r_6	r_15	10
		70 %	r_10	r_12	r_15	r_14	r_17	r_16	r_14	10
1		30 %	r_14	r_1	r_4	r_12	r_15	r_15	r_1	10
		50 %	r_16	r_11	r_11	r_15	r_14	r_16	r_14	10
		70 %	r_14	r_12	r_13	r_11	r_0	r_9	r_1	10
2		30 %	r_14	r_16	r_17	r_15	r_10	r_0	r_14	10
		50 %	r_14	r_7	r_14	r_15	r_12	r_11	r_14	10
		70 %	r_16	r_2	r_15	r_11	r_11	r_13	r_14	10

6.3.2.3 Results discussion

The allocation behaviour analysis and discussion will be done primarily on the allocation behaviour results from the experiments done on the specialized datasets (***Table 48***). These two (2) experiment results are chosen over the *averages only* results because the behaviour on the specialized dataset is easier to analyse. However, the same findings can be applied to the results from ***Table 47***. The overview of resource classification, as given in the introduction of this result subsection, is summarized in the table below:

Table 49 – Resource performance classification

Average Performers	Throughput Time Specialized	Cost Specialized
r_0 – r_9	**r_10 – r_13**	**r_14 – r_17**

The main results and conclusion which can be drawn from these specific results is that there are several signs indicating that the optimizer has an "understanding" of the behaviours of its resources in relation to the activities they are being assigned to. This understanding is primarily drawn from the behaviour exhibited on changing the objective mode, so switching between which objective is being compromised and which objective is being optimized, and on changing the compromise allowance.

The "understanding" of resources in relation to the given configuration can for example be observed in tree version one (1) of the cost objective constrained setup in ***Table 48*** (the second to last three (3) results). When the compromise allowance is changed, the class of resources also changes. It changes from resources specialized in cost performance to resources more focussed on time performance. The table below shows the changing resource population for this example.

Table 50 – Resource class shift overview between three (3) compromise allowances.

Compromise Allowance	Average Performers Count	Throughput Time Specialized Count	Cost Specialized Count
30 %	3 (42.86 %)	1 (14.29 %)	3 (42.86 %)
50 %	0 (0.00 %)	2 (28.57 %)	5 (71.43 %)
70 %	3 (42.86 %)	3 (42.86 %)	1 (14.29 %)

As can be seen, when the compromise allowance for the cost objective increases, so the cost objective is given lower priority, the number of throughput time resources increases. Relating this same case to the results in ***Table 46*** this allocation behaviour of switching from cost specialized resources to throughput time specialized resources can also be seen in the percentual improvement differences. As more throughput time resources are used, and less cost resources, the throughput time performance increases and the cost performances decreases.

7 Discussion: Limitations and opportunities

Whilst the research, design and experimentation have been performed with a high regard for the scientific methods, contribution-orientated research, thoroughness, consistency, transparency, and repeatability in mind, this thesis is not perfect and exhaustive in this topic. Unfortunately, due to scoping, or other, limitations, some elements of this research have only been touched on briefly or have been executed unfavourably. These limitations will be covered in this section.

Besides the limitations, or sometimes even alongside the limitations, opportunities stemming from this thesis will also be discussed. This resource allocation optimizer, based on a divide and conquer approach, has not only developed several interesting conclusions, but it also created several opportunities for future research.

7.1.1 Experiment data

The experiment data of this research is pseudorandom generated data. This data has been generated to test and compare different aspects of the process optimizer. Whilst constructing data according to models has the benefit that the data can be tailormade to stress certain aspects of the process optimizer, it can also lead to strongly favoured results. When generating data specifically for this process optimizer, the data can be engineered in order to get results which favour a hypothesis. In this research this has happened to some degree, albeit unintentionally. A specific example is the use of "specialized resources" which are optimized for one (1) objective, whilst it compromises on the other objective. This setup led to favourable results in an attempt to show the existence of a Pareto optimum front (see **Multi-objective optimization applicability**) in the use case of this optimizer. Whilst compromises on objectives in real world applications are not unthinkable, this should still be mentioned. This example is likely not the only example, as there might be other unintentional favouring of results.

Another potential issue with the used data is that it is not real. Whilst the data has been engineered to somewhat model and mirror real-world scenarios, it is still pseudorandom data based on models. Real-world data might have made the results in this thesis more relevant. The primary reason why real-world data has not been used is mostly due to unavailability. Either the datasets were too small in order to perform enough experiments on them, or the data lacked too many of the properties for which the requirements are set out in **Optimizer data requirement**.

7.1.2 Multi-objective optimizations & Genetic Algorithms

The implementation of the multi-objective optimizer in this thesis, which uses the ε-constraint method, only presents one (1) solution with a compromise on both (or rather all) of the objectives. Whilst this is an improvement on single-objective optimizations, as it could be theorized that processes and/or supply chains rarely ever have one (1) single objective, it still does not offer a desired result. A more favourable result for the multi-objective optimizer is that the optimizer gives the user a spectrum of solutions on which objectives are compromised more on or less on. This gives the user a less arbitrary choice of choosing their desired compromise, as the result of their compromise decision is clearly visible, as compared to how the optimizer works now, where a certain compromise allowance has to be chosen beforehand without knowing how this affects the objectives.

Multi-objective optimization techniques gave a possible solution for this single point solution issue: Genetic multi-objective optimization algorithms. Genetic multi-objective optimization algorithms do not work on a single solution, but instead work on a set of solutions. A conceivable follow up research could be a study into the applicability of genetic algorithms into this specific type of optimization problem.

7.1.3 Optimizer framework choice, techniques & Algorithms

The optimizer framework used in the prototype implementation for this thesis is the IBM-CPLEX optimizer. This optimizer was used for its ease of use in modelling and because of its accessibility thanks to the academic licences which are available. A shortcoming is that there is no comparison or otherwise consideration for other optimization frameworks, such as but not limited to Gurobi. However, it should

be noted that the comparison of performance or "ease of use" of optimizer frameworks is not the focus of this thesis and would be entirely outside of the scope of this thesis.

Continuing on about the choice of optimizers: This thesis uses the constraint programming optimizer from the CPLEX optimizer. The CPLEX optimizer uses generalized techniques and models to optimize a given problem. However, using an optimizer framework is not the only method of optimizing a problem. There are many different optimization algorithms, some of which could be better suited for an optimization problem such as the one presented in this thesis. This has unfortunately not been explored.

There are three (3) follow up studies which can be performed based on this limitation. The first two (2) studies are smaller, and the third study is a larger study. The first, smaller, follow-up study is to compare different optimization frameworks, as well as their modelling approach, and see if there is a more suitable framework available.

A second, smaller, study can be focussed on hyper parameter and configuration tuning of the optimizer. Currently the CPLEX optimizer is used as is, without any hyper parameter tuning. Adjusting the hyper parameters might yield better results.

The third study, which is a more substantial study, is to examine different optimization algorithms, including algorithms and techniques for, for example, constraint programming and mathematical programming, and try to evaluate which (type of) algorithm fits best to this specific optimization problem.

7.1.4 Cost Modelling

The optimization method in this thesis uses regression models to construct a cost model for each objective for each activity-resource pair. Regression models have been used as they seemed to be an easy to use, approachable, and most importantly well-fitting modelling candidate for the described problem. There are however a few shortcomings in the modelling setup.

The first shortcoming is the lack, or exclusion, of other variables. Currently only two (2) input variables are considered, activity occurrence count and resource allocation count. In a real-world application, there could be more than just two (2) variables which impact the performance of a certain activity-resource pairing. This thesis, for sake of simplicity, has not explored the inclusion of other variables, even though they might make the models more powerful and accurate.

In future research multiple per model and/or overarching variables could be considered. Examples of these model extensions could be: weather, temperature, season, traffic conditions, etc. Some of these variables could be per model specific, others could be applied globally.

The second shortcoming is the lack of comparison and consideration of different modelling techniques. Whilst the currently proposed regression models, linear and polynomial models, seem to model well within the provided data and scenarios, there are other regression techniques and other modelling techniques. Some of these different modelling techniques might provide a more accurate, or better fitting, model for the given data.

7.1.5 Objective function expansion

Related to the extension of the cost modelling techniques, objective functions could also be extended and expanded. This thesis only considers three (3) objective functions, and only two (2) of these objective functions have been implemented. However, it's likely that the three (3) proposed objective functions are not exhaustive for every (business) process and/or supply chain. Some processes might have objectives such as climate impact (CO2 impact), customer satisfaction, production capacity, production and/or product quality, etc. A follow-up study could focus on formulating a method of how to define and formalize (with formulae and functions) objective functions. When defining and formalizing objective functions, one should for example consider how to model these functions for the cost models, and how to model these functions for the optimizer.

7.1.6 Optimizer comparison

The optimizer has been shown to be successful on improving against the average baseline on test data, as can be seen in **Difference against baseline performance**. However, something which remains unknown is how “impressive” or “significant” these results are. This thesis does not compare this optimizer against other process optimizer, so there is no baseline comparison and improvement analysis. This lack of comparison mostly comes forth from the lack of availability of similar process optimizers to compare against.

7.1.7 Divide and conquer

For the last limitation, the optimizer in this thesis uses a form of divide and conquer when trying to optimize a process. This is done by first optimizing variants of a process and then merging these variants into a general solution for a process. The limitation is in the merging of the divide and conquer results.

Whilst the divide and conquer method used in this thesis seemed to be effective enough, the final solution merging method appears to be lacking. The merging methods are able to merge solutions into a general solution, but they only take simple statistics into account. Therefor a study could be dedicated to the solution merging methods. The solution merging methods have been argued to be lacking because they might put too much weight on larger variants, which do not result in a majority vote, or because they do not take overall allocation effects into account. As the merging step is the final step which converts optimal solutions of the variants into a general solution, it can have a large impact on the performance of the optimized result.

8 Conclusions

The introduction of this thesis posed two (2) research questions. Throughout this thesis, several steps have been taken, and methods have been developed and compared to answer the two (2) research questions.

The first of the two (2) research questions was formulated as follows:

> *"What are the (minimum) data requirements, in terms of format, contents, and quantity, in order to allow an optimizer to perform resource-activity allocation optimizations?"*

This question has primarily been answered in **Data Requirements**. The format was defined in the following manner, a process trace should have at least the following fields so that the process optimizer can translate a trace into a process tree and link resources with activities:

- **CASE IDENTIFIER**;
- **ACTIVITY IDENTIFIERS**;
- **START TIMESTAMP**;
- **RESOURCE IDENTIFIERS**.

Additionally, depending on the used objectives, several other fields data columns have to be added to a process trace as well. This thesis defined three (3) objectives, throughput time, wait time and cost, and the data columns for those objectives would be as follows:

- **ACTIVITY EXECUTION DURATION TIME**;
- **ACTIVITY WAIT TIME**;
- *(Optional)* **END TIMESTAMP**;
- **ACTIVITY EXECUTION COST**.

Note that these data column requirements are specific for the objectives defined in this thesis. Different objective functions might bring different data column requirements with them. The data format requirements are therefore partially dependent on the scenario in which the optimizer is to be applied.

The second part of this research question was about data quantity. The data quantity can be split up in two (2) parts, the first being for the process miner, to construct a process tree out of a trace, and the second being for the cost modellers. The first, the process miner, does not have strict minimum number of cases it requires, as it heavily depends on the complexity of the process for how many cases are required. Rather, enough cases should be provided to present a representative set of process variants. The second question regarding the cost modelling is simpler and has statistical backing through other, previous, studies. Ideally for every resource-activity cost model, there should be a minimum of between ten (10) to 25 cases in order to create a representative enough model. The complexity of the underlying model, i.e. being more linear or more polynomial, could sway the quantity requirement to either of the two (2) aforementioned minima. Note that a scenario could arise where resource-activity pairs may share cases, so five resource-activity pair models, do not necessarily need 125 cases to create accurate model.

The second question posed in the introduction was:

> *"How can process mining be used to optimize the allocation of resources to activities, using a divided and conquer approach?"*

This question has been approached and answered primarily in **Method & Design**. The answer is given as an optimization method where the provided process, in the form of a process trace data-set, is converted into a process tree using process mining, which is then optimized in a divide and conquer style. This divide and conquer style works by optimizing variants of the process, and then merging the

results. The optimizations are done by defining the objectives as objective functions using objective cost modelling functions and subjecting them to a set of constraints. The optimization output is a set of recommended activity-resource combinations.

The optimizer has been tested and proven in potential (see **Difference against baseline performance**) by using self-generated simulated data. This simulated data, as described in **Experiment Data**, allowed to gain five main insights into the optimizer. Firstly, it showed the difference between the two (2) modelling techniques, linear models and polynomial models. It showed the performance differences and it allowed for a trade-off analysis between the two (2) methods. Secondly, it showed the advantages and the validity of a divide and conquer approach with experiments on the variant selection methods. Using the simulated data, the pruning of variants was shown to improve performance fairly substantially on runtime whilst only showing a limited impact on improvement performance. Thirdly, it laid bare an improvement area of the optimizer in the solution merging methods. Fourthly, data could be generated to specifically stress the multi-objective optimization component of the optimizer. Not only was a Pareto optimum shown to exist in an example process, as seen in **Multi-objective optimization applicability**, the optimizer also had also shown the ability to deal with multi-objective optimizations effectively. And lastly, tying into the last point, because the behaviour of resources has been pre-defined, it was possible to create an insight into the allocation behaviour of the optimizer. The optimizer has shown to "*understand*" the behaviour of certain resources and it is able to differentiate between resources based on their "*specialization*" when shifting the optimization priority between objective functions, as can be seen in **Allocation behaviour**.

Perhaps a more important question to answer is what a reader can learn from reading this thesis. As mentioned in the introduction of this thesis and in the conclusion, this thesis describes how to construct a process optimizer with the focus on multi-objective resource allocation optimizations using process mining techniques. In describing this, a framework is built which allows users and researchers to develop several insights, such as process behaviour and complex resource behaviour, by just analysing and mining a process log. This zero-knowledge basis, the notion that the optimizer should gather all its insights just from a process log using process mining techniques and not from any other external sources, is a fairly unique characteristic of this particular optimizer when compared to existing resource allocation optimizers. Other resource allocation optimizers require pre-defined resource behaviour models to be given.

Another contribution which a reader should extract from this thesis is a method of expandability of this framework. Not only is a framework provided which shows how to optimize resource allocations by means of only reading and analysing logs, several descriptions and methods have also been provided on how this framework could be made more advanced. This thesis for example describes how more objectives can be added, how more performance impacting variables on resource behaviour can be added (examples are weather, climate, rain, employee satisfaction, season, etc.), and how more objective constraints can be added. Thereby this thesis presents an optimization framework truthful to the definition of a "*framework*", and it should therefor offer several new research opportunities in the field of resource allocation optimizations and process mining.

Referring back to the results in **Results** and to the found knowledge gaps in **Knowledge gap**, several findings can be added to the conclusions above:

- **DIVIDE AND CONQUER** This thesis has shown that the divide and conquer technique, together with the merging technique, is a successful candidate for resource allocation optimizations. Not only can a stochastic process be converted into deterministic variants, they can also be individually optimized, and their individual results can be merged to serve as a generalizable solution for the process. Another finding in the divide and conquer approach is that not every variant of a process has to be optimized. This means that a complex stochastic model with many

different variants can essentially be optimized by only focussing on a (small) subset of process variants, which potentially decreases the required runtime significantly;

- **DYNAMIC MODELS** A second finding is that dynamic models, based on regression on the available process data, can successfully be used as cost models in a resource allocation optimization exercise. This has two (2) implications. The first being that optimizing the allocation of resources no longer requires an (a-priori) in-depth knowledge of the behaviour of resources, which has been used as the theoretical basis for several other resource allocation techniques. The second implication is that, because the dynamic models are regression models, these models can easily be expanded with more inputs using existing regression theories and techniques. So, taking external factors/parameters into regard when wanting to optimize a process, such as weather, seasons, etc. does not require a complete redesign of the optimizer. Instead, the user can simply rely on basic regression theories and include these extra parameters into the regression-based cost models;
- **POLYNOMIAL APPLICABILITY** Extending the finding on the dynamic models, not only was it shown that the optimizer works with linear models, it has also been shown that the optimizer can work with polynomial regression models. In fact, not only are polynomial models applicable, but they also provide a higher overall accuracy. The primary implication which can be drawn from this, is that the optimizer can be used in more complex environments, where it is not known whether the behaviour of resources exhibit themselves in a linear fashion, or in any other predefined curve shape;
- **MODEL COMPLEXITY** The last finding is that, thanks to the removal of the stochastic elements and the applicability of linear cost models, the optimizer is shown to have the potential to operate in a lower complexity category. To be specific, theoretically, this optimizer, and therefore resource allocation optimizations, could operate with NP-complete complexity, instead of NP-hard complexity according to *Karp's 21 NP-Complete problems* [30].

9 References

[1] W. M. P. van der Aalst, “Process Mining in the Large: A Tutorial,” *Springer International Publishing Switzerland* , vol. BISS 2013, pp. 33–76, 2014, doi: 10.1007/978-3-319-05461-2.

[2] W. van der Aalst, T. Weijters, and L. Maruster, “Workflow Mining: Discovering Process Models from Event Logs,” 2004.

[3] W. van der Aalst, “Process mining: Overview and opportunities,” *ACM Transactions on Management Information Systems*, vol. 3, no. 2. Jul. 2012. doi: 10.1145/2229156.2229157.

[4] W. M. P. van der Aalst, “Business Process Management: A Comprehensive Survey,” *ISRN Software Engineering*, vol. 2013, pp. 1–37, Feb. 2013, doi: 10.1155/2013/507984.

[5] M. Siek and R. M. G. Mukti, “Business process mining from e-commerce event web logs: Conformance checking and bottleneck identification,” in *IOP Conference Series: Earth and Environmental Science*, Apr. 2021, vol. 729, no. 1. doi: 10.1088/1755-1315/729/1/012133.

[6] R. Bemthuis, N. van Slooten, J. J. Arachchige, J. P. S. Piest, and F. A. Bukhsh, “A classification of process mining bottleneck analysis techniques for operational support,” in *Proceedings of the 18th International Conference on e-Business, ICE-B 2021*, 2021, pp. 127–135. doi: 10.5220/0010578601270135.

[7] G. Gill and M. Singh, “Bottleneck analysis and alleviation in pipelined systems: A fast hierarchical approach,” in *Proceedings - International Symposium on Asynchronous Circuits and Systems*, 2009, pp. 195–205. doi: 10.1109/ASYNC.2009.20.

[8] A. Barone, “Bottleneck,” *https://www.investopedia.com/terms/b/bottleneck.asp*, Jul. 2022.

[9] Iso, “THE PROCESS APPROACH IN ISO 9001:2015,” 2015. [Online]. Available: www.iso.org

[10] A. Berti, S. J. van Zelst, and W. M. P. van der Aalst, “Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science.” [Online]. Available: http://python.org

[11] W. van der Aalst *et al.*, “PM4PY,” *https://pm4py.fit.fraunhofer.de*.

[12] W. van der Aalst, T. Weijters, and L. Maruster, “Workflow Mining: Discovering Process Models from Event Logs,” 2004.

[13] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. Alves De Medeiros, “Process Mining with the HeuristicsMiner Algorithm.”

[14] W. van de Aalst, “Process discovery: Capturing the invisible,” *IEEE Comput Intell Mag*, vol. 5, no. 1, pp. 28–41, Feb. 2010, doi: 10.1109/MCI.2009.935307.

[15] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “LNCS 7927 - Discovering Block-Structured Process Models from Event Logs - A Constructive Approach.”

[16] W. M. P. van der Aalst, “Business Process Management: A Comprehensive Survey,” *ISRN Software Engineering*, vol. 2013, pp. 1–37, Feb. 2013, doi: 10.1155/2013/507984.

[17] M. Arias, E. Rojas, J. Munoz-Gama, and M. Sepúlveda, *Business Process Management Workshops*, vol. 256. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-42887-1.

[18] X. Jiajie, L. Chengfei, and Z. Ziaohui, “Resource Allocation vs. Business Process Improvement: How They Impact on Each Other,” Melbourne, 2008.

[19] M. Arias, E. Rojas, J. Munoz-Gama, and M. Sepúlveda, *Business Process Management Workshops*, vol. 256. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-42887-1.

[20] W. Zhao, L. Yang, H. Liu, and R. Wu, "The optimization of resource allocation based on process mining," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015, vol. 9227, pp. 341–353. doi: 10.1007/978-3-319-22053-6_38.

[21] Z. Huang, W. M. P. van der Aalst, X. Lu, and H. Duan, "Reinforcement learning based resource allocation in business process management," *Data Knowl Eng*, vol. 70, no. 1, pp. 127–145, Jan. 2011, doi: 10.1016/j.datak.2010.09.002.

[22] P. Korhonen and M. Syrjänen, "Resource allocation based on efficiency analysis," *Manage Sci*, vol. 50, no. 8, pp. 1134–1144, 2004, doi: 10.1287/mnsc.1040.0244.

[23] W. Mckinney, "Data Structures for Statistical Computing in Python," 2010.

[24] The pandas development team, "pandas-dev/pandas: Pandas." Zenodo, Jun. 23, 2022.

[25] R. M. Yerkes A N D J O H N and D. Dodson, "THE RELATION OF STRENGTH OF STIMULUS TO RAPIDITY OF HABIT-FORMATION."

[26] X. Xu, Y. Wang, M. Li, and H. K. Kwan, "Paradoxical Effects of Performance Pressure on Employees' In-Role Behaviors: An Approach/Avoidance Model," *Front Psychol*, vol. 12, Oct. 2021, doi: 10.3389/fpsyg.2021.744404.

[27] J. Hofmans, J. Debusscher, E. Dóci, A. Spanouli, and F. de Fruyt, "The curvilinear relationship between work pressure and momentary task performance: The role of state and trait core self-evaluations," *Front Psychol*, vol. 6, no. OCT, 2015, doi: 10.3389/fpsyg.2015.01680.

[28] C. Tardi, M. Cheng, and T. Li, "80-20 Rule Definition," 2022. https://www.investopedia.com/terms/1/80-20-rule.asp (accessed Aug. 08, 2022).

[29] S. G. Johnson, "A Brief Overview of Optimization Problems," 2008.

[30] R. M. Karp, "Reducibility Among Combinatorial Problems," in *50 Years of Integer Programming 1958-2008*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 219–241. doi: 10.1007/978-3-540-68279-0_8.

[31] L. C. Coelho, "How to linearize max, min, and abs functions." https://www.leandro-coelho.com/how-to-linearize-max-min-and-abs-functions/ (accessed Aug. 04, 2022).

[32] L.-C. Kung, "Operations Research Applications of Linear Programming."

[33] IBM, "IBM ILOG CPLEX Optimization Studio CPLEX User's Manual."

[34] L. Gurobi Optimization, "GUROBI OPTIMIZER REFERENCE MANUAL."

[35] IBM, "Mathematical programming versus constraint programming." http://ibmdecisionoptimization.github.io/docplex-doc/mp_vs_cp.html (accessed Aug. 04, 2022).

[36] R. Sridharan, "Statistics for Research Projects," 2019.

[37] F. Pedregosa FABIANPEDREGOSA *et al.*, "Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot," 2011. [Online]. Available: http://scikit-learn.sourceforge.net.

[38] A. E. Smith, M. Gulsen, and D. M. Tate, “A genetic algorithm approach to curve fitting,” *Int J Prod Res*, vol. 33, no. 7, pp. 1911–1923, 1995, doi: 10.1080/00207549508904789.

[39] P. Sinha, “Multivariate Polynomial Regression in Data Mining: Methodology, Problems and Solutions,” *Int J Sci Eng Res*, vol. 4, 2013, [Online]. Available: http://www.ijser.org

[40] I. H. Witten, E. Frank, and M. A. Hall, “Training and Testing,” in *Data Mining Practical Machine Learning Tools and Techniques*, Third., Elsevier, 2011, pp. 148–150.

[41] I. H. Witten, E. Frank, and M. A. Hall, “coefficient of determination,” in *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed., Elsevier, 2011, pp. 180–180. doi: 10.1016/C2009-0-19715-5.

[42] I. H. Witten, E. Frank, and M. A. Hall, “Root Mean Square Error,” in *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed., Elsevier, 2011, pp. 180–180. doi: 10.1016/C2009-0-19715-5.

[43] H. Pishro-Nik, “Probability Density Function,” in *Introduction to Probability, Statistics, and Random Processes*, Kappa Research, 2014, pp. 163–167.

[44] S. Sudhoff, “Multi-Objective Optimization.”

[45] K. Deb, “Weighted Sum Method,” in *Multi-Objective Optimization Using Evolutionary Algorithms*, 1st ed., vol. 1, S. Ross and R. Weber, Eds. Chichester, New York, Weinheim, Drisbane, Singapore, Toronto: John Wiley & Sons, Ltd., 2001, pp. 50–56.

[46] K. Deb, “ε-Constraint Method,” in *Multi-Objective Optimization Using Evolutionary Algorithms*, 1st ed., vol. 1, S. Ross and R. Weber, Eds. Chichester, New York, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Ltd., 2001, pp. 57–60.

[47] K. Deb, “Weighted Metric Method,” in *Multi-Objective Optimization Using Evolutionary Algorithms*, 1st ed., vol. 1, S. Ross and R. Weber, Eds. Chichester, New York, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Ltd., 2001, pp. 60–65.

[48] I. Giagkiozis and P. J. Fleming, “Methods for multi-objective optimization: An analysis,” *Inf Sci (N Y)*, vol. 293, pp. 338–350, Feb. 2015, doi: 10.1016/j.ins.2014.08.071.

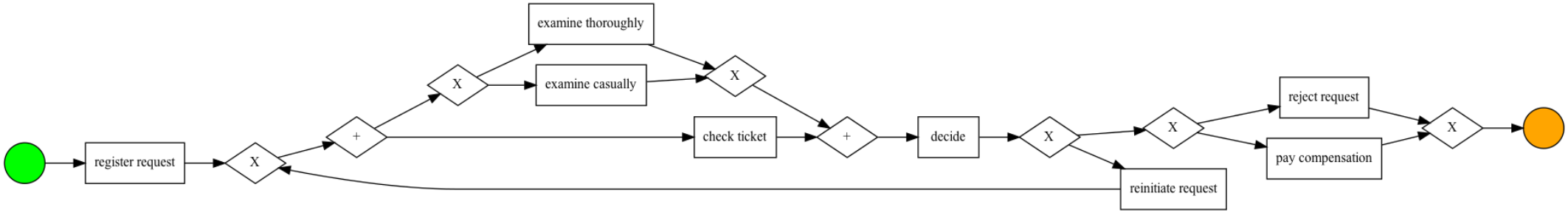
[49] Fluxicon and W. van der Aalst, “Process Mining Book,” *https://fluxicon.com/book/read/dataext/*, 2022.

[50] D. G. Jenkins and P. F. Quintana-Ascencio, “A solution to minimum sample size for regressions,” *PLoS One*, vol. 15, no. 2, Feb. 2020, doi: 10.1371/journal.pone.0229345.

[51] J. F. Hair JR., W. C. Black, B. J. Babin, and R. E. Anderson, “Regression requirements,” in *Multivariate Data Analysis*, 7th ed., Pearson, 2014, pp. 573–574.

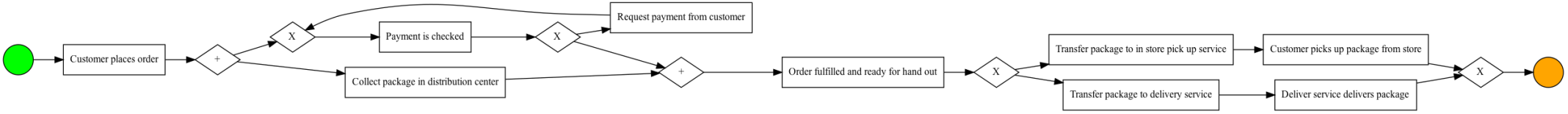
10 Appendices

10.1 Process model example 0, represented in a BPMN schema



Process based on a dataset example curtesy of PM4PY [11].

10.2 Process model example 1, represented in a BPMN schema



10.3 Process model example 1, complete example event-data trace 1

Case	Activity	Resource	Cost	Timestamp Start
0	Customer places order	-	0	12:00:00
0	Payment is checked	Erik	11	12:00:10
0	Collect package in distribution center	Jim	82	12:00:10
0	Order fulfilled and ready for hand out	Pete	33	12:09:32
0	Transfer package to in store pick up service	Jim	20.5	12:12:41
0	Customer picks up package from store	-	10	12:15:01
1	Customer places order	-	0	18:00:00
1	Collect package in distribution center	Mike	74.4	18:00:10
1	Payment is checked	Erik	10.25	18:00:10
1	Request payment from customer	Erik	51.25	18:02:40
1	Payment is checked	Erik	10.25	18:15:10
1	Order fulfilled and ready for hand out	Ingrid	27	18:17:40
1	Transfer package to in store pick up service	Kim	18.8	18:21:31
1	Customer picks up package from store	-	10	18:23:25
2	Customer places order	-	0	02:00:00
2	Payment is checked	Erik	11	02:00:10
2	Collect package in distribution center	Jim	90	02:00:10
2	Order fulfilled and ready for hand out	Ingrid	27	02:08:34
2	Transfer package to delivery service	Mohammed	28.5	02:12:25
2	Deliver service delivers package	Chelsea	78.4	02:15:21
3	Customer places order	-	0	08:00:00
3	Payment is checked	Shenna	9.5	08:00:10
3	Collect package in distribution center	Jim	90	08:00:10
3	Request payment from customer	Erik	55	08:01:58
3	Payment is checked	Shenna	9.5	08:12:58
3	Order fulfilled and ready for hand out	Pete	33	08:14:46
3	Transfer package to delivery service	Tim	22.5	08:17:55
3	Deliver service delivers package	Tim	60	08:21:20
4	Customer places order	-	0	16:00:00
4	Payment is checked	Shenna	9.5	16:00:10
4	Collect package in distribution center	Jim	82	16:00:10
4	Order fulfilled and ready for hand out	Ingrid	27	16:09:32
4	Transfer package to in store pick up service	Jim	20.5	16:13:23
4	Customer picks up package from store	-	10	16:15:43
5	Customer places order	-	0	17:00:00
5	Collect package in distribution center	Jim	82	17:00:10
5	Payment is checked	Erik	11	17:00:10
5	Order fulfilled and ready for hand out	Ingrid	27	17:09:32
5	Transfer package to in store pick up service	Jim	20.5	17:13:23
5	Customer picks up package from store	-	10	17:15:43

10.4 Process model example 1, complete resource-activity allocation map for complete example event-data trace 1

	Activity	Resource
0	Customer places order	-
1	Payment is checked	Erik
2	Payment is checked	Shenna
3	Collect package in distribution center	Jim
4	Collect package in distribution center	Mike
5	Order fulfilled and ready for hand out	Pete
6	Order fulfilled and ready for hand out	Ingrid
7	Request payment from customer	Erik
8	Transfer package to delivery service	Mohammed
9	Transfer package to delivery service	Tim
10	Transfer package to in store pick up service	Jim
11	Transfer package to in store pick up service	Kim
12	Deliver service delivers package	Chelsea
13	Deliver service delivers package	Tim
14	Customer picks up package from store	-

10.5 Process model example 1, complete resource allocation limit for complete example event-data trace 1

Resource	Allocation limit
Erik	2
Shenna	1
Jim	2
Mike	1
Kim	1
Pete	1
Ingrid	1
Mohammed	1
Chelsea	1
Tim	2
-	2

10.6Linearization of a multi-element Max component

Suppose the following function:

$$x = \text{Max}\{x_0, x_1, x_2, x_3, x_4\}$$

The linearization of this multi element Max function can be achieved by dividing the problem into multiple two element Max functions and substituting these functions into the original problem.

$$M_0 = \text{Max}\{x_0, x_1\}$$

$$M_1 = \text{Max}\{M_0, x_2\}$$

$$M_2 = \text{Max}\{M_1, x_3\}$$

$$x = \text{Max}\{M_2, x_4\}$$

Now suppose a constant C such that $C \geq x_0, x_1, x_2, x_3, x_4$.

Next define a set of binary variables $\{\gamma_0, \gamma_1, \dots, \gamma_{n-1}\}$ such that $\gamma = \{1: a < b | 0: a \geq b\}$

$$\gamma_0 = \begin{cases} 1, & x_1 < x_0 \\ 0, & x_1 \geq x_0 \end{cases}$$

$$\gamma_1 = \begin{cases} 1, & x_2 < M_0 \\ 0, & x_2 \geq M_0 \end{cases}$$

$$\gamma_2 = \begin{cases} 1, & x_3 < M_1 \\ 0, & x_3 \geq M_1 \end{cases}$$

$$\gamma_3 = \begin{cases} 1, & x_4 < M_2 \\ 0, & x_4 \geq M_2 \end{cases}$$

Finally define a set of constraints such that $x \geq x_0, x_1, x_2, x_3, x_4$

$$x \geq x_0$$

$$x \geq x_1$$

$$x \geq x_2$$

$$x \geq x_3$$

$$x \geq x_4$$

$$x \leq x_0 + C \cdot (1 - \gamma_0)$$

$$x \leq x_1 + C \cdot \gamma_0$$

$$x \leq M_0 + C \cdot (1 - \gamma_1)$$

$$x \leq x_2 + C \cdot \gamma_1$$

$$x \leq M_1 + C \cdot (1 - \gamma_2)$$

$$x \leq x_3 + C \cdot \gamma_2$$

$$x \leq M_2 + C \cdot (1 - \gamma_3)$$

$$x \leq x_3 + C \cdot \gamma_3$$

10.7 Pareto Front Multi Objective Problem Space Setup

10.7.1 Process tree setup

This tree will only include loops as a special tree structure besides sequential structures. Parallel and XOR-branches are not included in this tree type. This tree should contain at a minimum three (3) activities, at a maximum six (6) activities, and should have a mode of four (4) activities. Furthermore, silent activities, which are empty activities used for more complex structures are omitted. The specific parameters used in the process trace generator for the tree generator are as follows:

Table 51 – Settings used for the loops only tree.

Seed 0	Seed 1	Min	Max	Mode	Silent (%)	Choice (%)	Parallel (%)	Loop (%)
4674	6666	3	6	4	0.0	0.0	0.0	25.0

The resulting tree contains one (1) loops consisting of one (1) base activity and one (1) loop activity, the loop component is preceded by two (2) activities in sequence, making a total of four (4) activities. A BPMN representation of the tree can be found in ***Figure 9***.

This tree is specifically generated to minimize the strain of exhaustive solution space generation by keeping the number of activities relatively low. Furthermore, a loop element has been specifically added in order to increase the variance in allocation results by means of re-occurrence and reallocation of activities and resources respectively, which are used in modifier functions (defined later on).

10.7.2 Simulation tree setup

The attachment of simulation values to the process tree is done with the following base settings:

Table 52 – Base settings for generating the Pareto tree simulation settings.

Max loop depth	Decrease loop probability	Even XOR probability
10	Yes	No

This setup has intentionally been chosen to maximize the loop size in order to increase the effect of the modifier functions.

The two seeds used are: 1660384210 for seed 0 and 1660384210 for seed 1. This results in the following simulation tree:

```
('>',
 ['a_b',
  ('>',
   ['a_c',
    ('*',
     [0.9159173561354, 0.6667583105152183, 0.1566936358192275,
      0.1307744020939576, 0.06640381641892078, 0.007841389604520596,
      0.004330130279075698, 0.0007035462527451663],
     ['a_a', 'a_d'])])])
```

Figure 25 – Sim tree for the Pareto tree.

10.7.3 Activity simulation values

All simulation values for the activities for all trees are generated with the same base parameters. The random seeds used for activity simulation values generation are the same seeds as the ones in the section above. The base parameters are set as follows:

Table 53 – Base settings for generating all activities.

Min Time	Max Time	Min Cost	Max Cost

10	1000	10	1000

For the re-occurrence functions, the following settings are used:

Table 54 – Modifier functions with re-occurrence of an activity (x) as input.

Time functions	
Function	**Probability distribution**
$f_{A_t}(x) = \frac{1}{2x} - \frac{1}{2}$	$\frac{2}{3}$
$f_{A_t}(x) = \sqrt[4]{x}$	$\frac{1}{3}$
Cost functions	
Function	**Probability distribution**
$f_{A_c}(x) = -\frac{x\sqrt[4]{x}}{20}$	$\frac{4}{5}$
$f_{A_c}(x) = -\frac{3x\sqrt{x}}{20}$	$\frac{1}{5}$

The settings in the tables above are explained in **Activity simulation values**.

Activities setup is performed with seed pair *1660384210* & *1660384210*, this results in the following setup:

Table 55 – Activities values for the Pareto process trace.

Activity	Base time value	Base cost value
a_a	271.845757	458.241301
a_b	565.749990	319.263352
a_c	733.317805	853.926122
a_d	889.181873	272.404352

10.7.4 Resource simulation values

The Pareto process should be a process where a Pareto optimum front can be formed based on solutions each with a unique combination of resources and activities. In order to create a Pareto optimum front, the activity resource combinations need to vary between effectiveness and ineffectiveness for the respective objectives; in this case, there need to be average performing combinations, throughput time effective and cost ineffective combinations, and cost effective and throughput time ineffective combinations.

In order to achieve the above, the following set of resources is proposed: A set of resources with both average performers, and two sets of specialized performers, one set who are cost effective and throughput time ineffective and one set who are cost ineffective and throughput time effective.

10.7.4.1 Base settings: Average performance resources

The **AVERAGE PERFORMANCE RESOURCES** resource generator for the Pareto process trace generator receives the following base settings:

Table 56 – Base parameter settings for average resources without random modifiers.

Min number of resources	Max number of resources	Jack of all trades penalty	Min time base modifier	Max time base modifier	Min cost base modifier	Max cost base modifier
2	2	0,25	-0,3	0,3	-0,2	0,2

Table 57 – Modifier functions with allocation of a resource x and a threshold value τ as input for average resources without random modifiers.

Time functions	
Function	**Threshold range**

$f_{R_t}(x,\tau) = \begin{cases} \frac{x}{2}, & x \le \tau \\ (x-\tau)^2 + \frac{\tau}{2}, & x > \tau \end{cases}$	D_τ: [3,7]
Cost functions	
Function	**Threshold range**
$f_{R_c}(x,\tau) = \begin{cases} -\frac{\sqrt{x}}{10}, & x \le \tau \\ -\frac{\sqrt{\tau}}{10}, & x > \tau \end{cases}$	D_τ: [5,10]

10.7.4.2 Base settings: throughput time specialized resources

The **THROUGHPUT TIME SPECIALIZED RESOURCES** resource generator receives the following base settings:

Table 58 – Base parameter settings for throughput time specialized resources.

Min number of resources	Max number of resources	Min time base modifier	Max time base modifier	Min cost base modifier	Max cost base modifier
2	2	-0.5	-0.5	0.5	0.5

The throughput time specialized resources have the following functions applied to them.

Table 59 – Modifier function for throughput time with allocation of a resource x as input for throughput time specialized resources.

Time function	Cost functions
$f_{R_t}(x) = \frac{\sqrt{x}}{10}$	$f_{R_c}(x) = -\frac{\sqrt{x}}{5}$

10.7.4.3 Base settings: cost specialized resources

The **COST SPECIALIZED RESOURCES** resource generator receives the following base settings:

Table 60 – Base parameter settings for cost specialized resources.

Min number of resources	Max number of resources	Min time base modifier	Max time base modifier	Min cost base modifier	Max cost base modifier
2	2	0.5	0.5	-0.5	-0.5

The cost specialized resources have the following functions applied to them.

Table 61 – Modifier function for cost with allocation of a resource x as input for throughput time specialized resources.

Time function	Cost functions
$f_{R_t}(x) = \frac{x^2}{10}$	$f_{R_c}(x) = -\frac{\sqrt{x}}{100}$

10.7.4.4 Resources setup

Resources setup is performed with seed pair *1660384210* & *1660384210*, this results in the following setup:

Table 62 – Average resources without random events and with specialized resources.

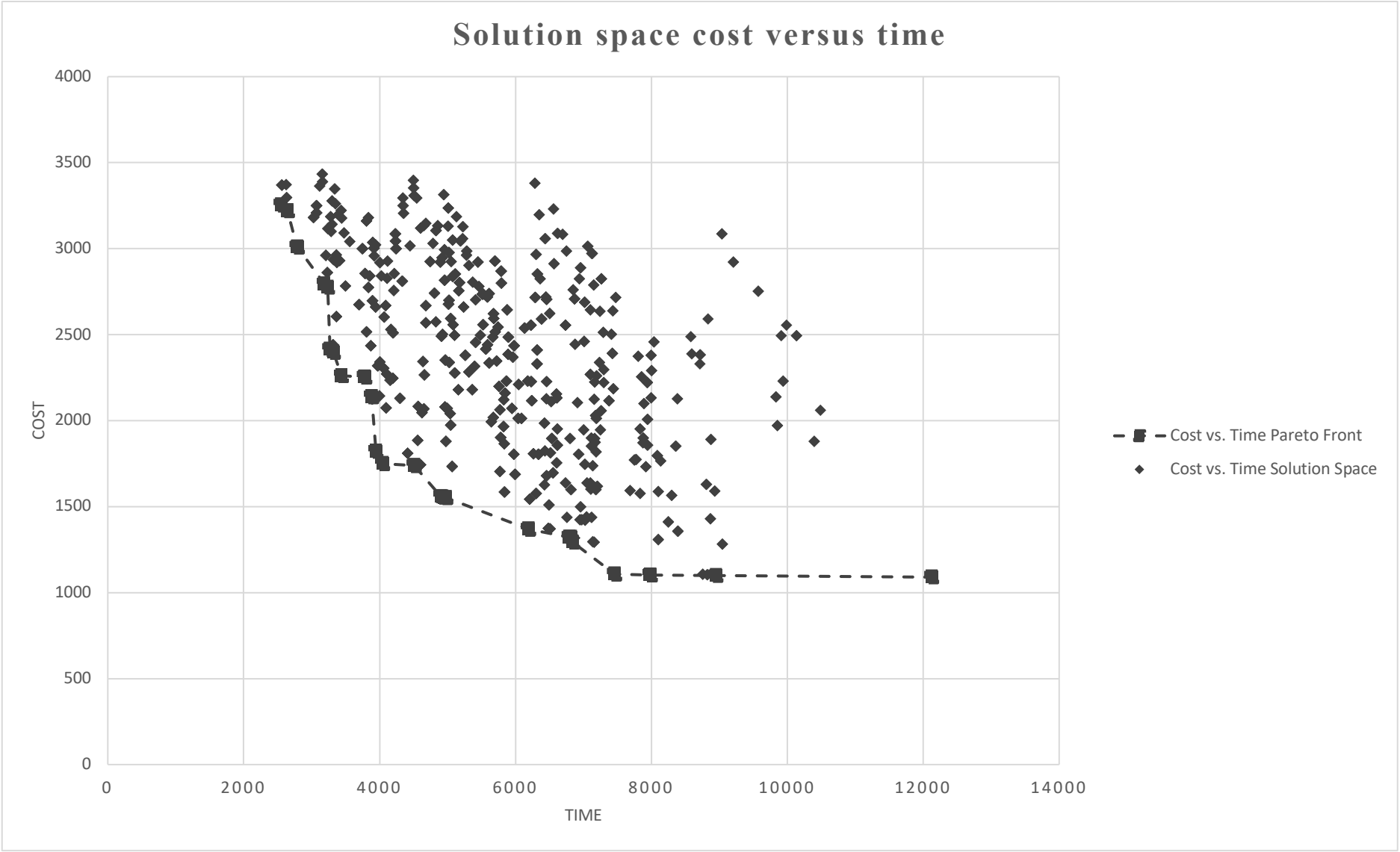
Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e	0.119357	0.109442	-0.029916	4	7
r_1	a_b, a_f, a_a, a_e, a_d	0.197911	0.195291	0.158918	3	7
Specialized resources						
r_2		0.222668	-0.500000	0.500000		
r_3		0.901155	-0.500000	0.500000		
r_4		0.171687	-0.500000	0.500000		

r_5		0.863877	-0.500000	0.500000		

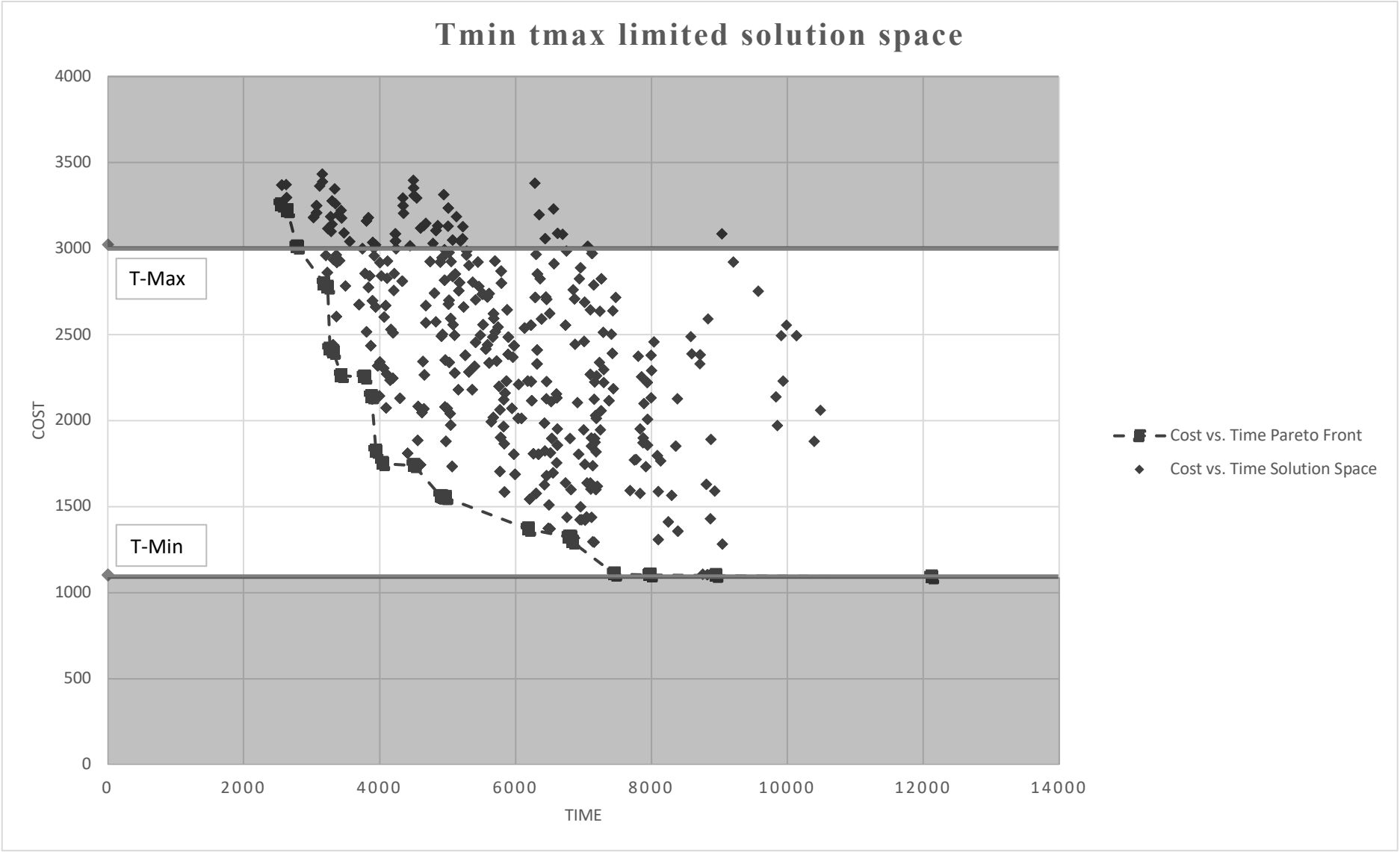
10.7.5 Case count

The pareto process trace will have 1.000 cases, produced by 1.000 simulation runs over all the above simulation values.

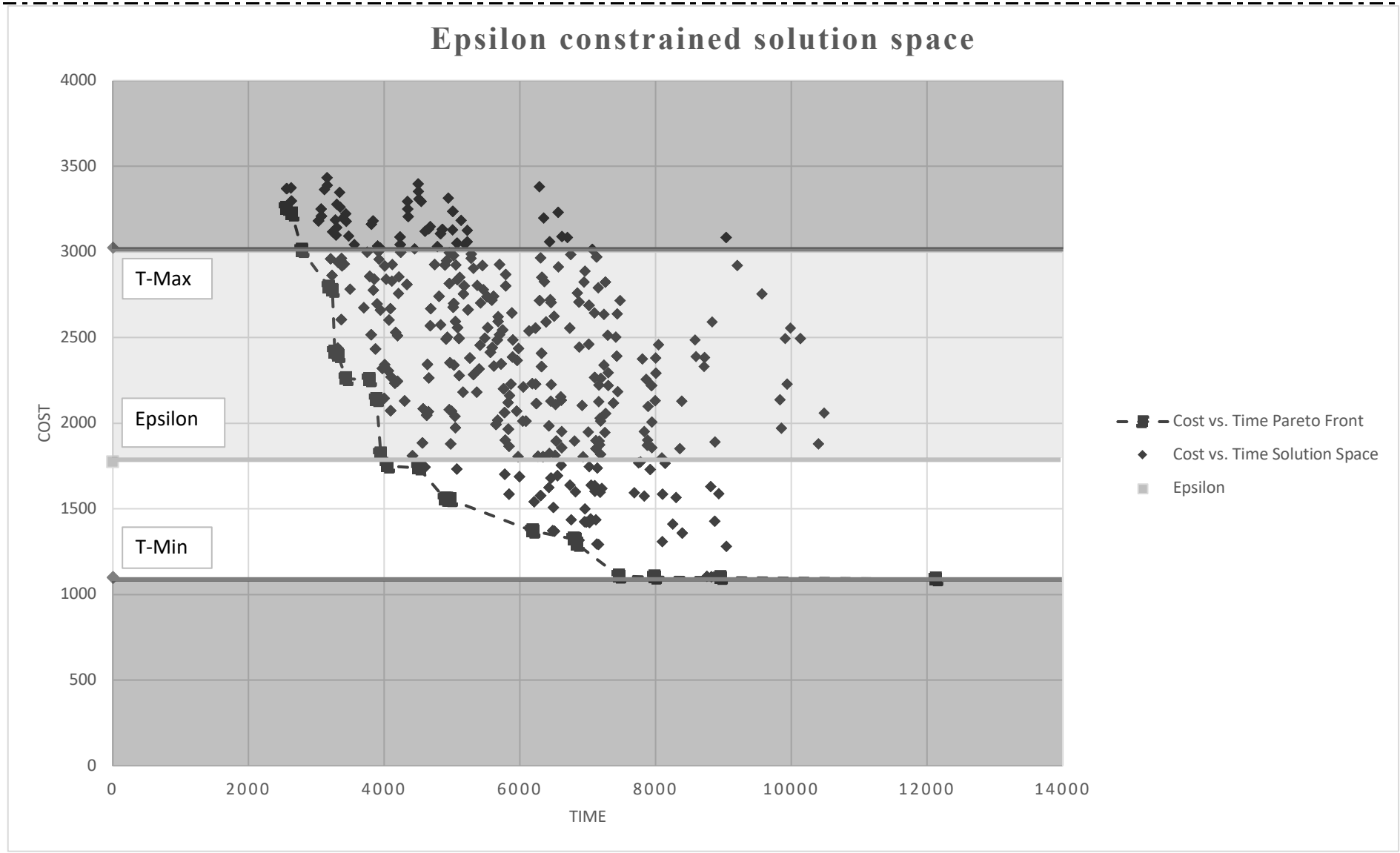
10.8Pareto solution space visualisation



10.9Tmin Tmax solution space visualisation



10.10 Epsilon constrained problem space visualisation



10.11 Process trace generator: Simulation tree results

10.11.1 Loops only

The three (3) seed combinations are:

Table 63 – Sim tree seeds for the loops only tree

	Sim Tree 0	Sim Tree 1	Sim Tree 2
Seed 0	827811881	78488783	751954824
Seed 1	892884050	571064686	113944411

The three (3) resulting trees with simulation values are as follows:

```
('>',
 [('>',
   [('*',
     [0.49858121831207936, 0.06278856931551752, 0.0066042507336793056],
     ['a_a', 'a_d']),
    ('*',
     [0.6012613504991797, 0.591790226193634, 0.4818295384953422,
      0.37817439420305476, 0.3486813944177145],
     ['a_f', 'a_c'])]),
  ('*',
   [0.8033917029517865, 0.40319111961786586, 0.388562850457725],
   ['a_e', 'a_b'])])
```

Figure 26 – Sim tree 0 of the loops only tree.

```
('>',
 [('>',
   [('*',
     [0.4158709049820133, 0.034698704354528355, 0.024575995808307036],
     ['a_a', 'a_d']),
    ('*',
     [0.2865720514227674, 0.2499755715447975, 0.12976540173076961,
      0.03248390518516331, 0.018702818046026887],
     ['a_f', 'a_c'])]),
  ('*',
   [0.12123977990767587,.0534741423865912, 0.047490588820646114,
    0.024722220073648046],
   ['a_e', 'a_b'])])
```

Figure 27 – Sim tree 1 of the loops only tree.

```
('>',
 [('>',
   [('*',
     [0.9437539084778306, 0.2749080246815939, 0.072500314790804],
     ['a_a', 'a_d']),
    ('*',
     [0.9629261359177992, 0.4066324434639764],
     ['a_f', 'a_c'])]),
  ('*',
   [0.10904550783693645, 0.06689673567119508, 0.06262154134423079,
    0.026706132876516357, 0.0027815165080447304],
   ['a_e', 'a_b'])])
```

Figure 28 – Sim tree 2 of the loops only tree.

10.11.2 Parallel branches only

The three (3) seed combinations are:

Table 64 – Sim tree seeds for the parallel branches tree.

	Sim Tree 0	Sim Tree 1	Sim Tree 2
Seed 0	1579143339	1216797035	1075399334
Seed 1	617776679	1143991442	1754616638

The three (3) resulting trees with simulation values are as follows:

```
('>',
 [('+',
  [('>',
   ['a_c', 'a_d']),
   ('>',
    ['a_b',
    ('+',
     ['a_e', 'a_f'])])]),
  ('>', ['a_g', 'a_a'])])
```

Figure 29 – Sim tree 0 of the parallel branches only tree.

The three (3) simulation tree outputs are identical. This is because parallel branches have no probability values to set. However, the three (3) simulation seed pairs will also be used for activity and resource model construction.

10.11.3 XOR-branches only

The three (3) seed combinations are:

Table 65 – Sim tree seeds for the XOR-branches tree.

	Sim Tree 0	Sim Tree 1	Sim Tree 2
Seed 0	1088179171	694198756	1150356829
Seed 1	28661790	640368126	1399333154

The three (3) resulting trees with simulation values are as follows:

```
('X',
 [0.2857142857142857, 0.7142857142857143],
 [('>',
   ['a_c',
    ('X',
     [0.42857142857142855, 0.5714285714285714],
     ['a_e',
     ('>',
      ['a_f', 'a_b'])])]),
  ('X',
   [0.16666666666666666, 0.8333333333333334],
   ['a_a', 'a_d'])])
```

Figure 30 – Sim tree 0 of the XOR-branches only tree.

```
('X',
 [0.3333333333333333, 0.6666666666666666],
 [('>',
   ['a_c',
    ('X',
     [0.75, 0.25],
     ['a_e',
     ('>',
      ['a_f', 'a_b'])])]),
  ('X',
   [0.5, 0.5],
   ['a_a', 'a_d'])])
```

Figure 31 – Sim tree 1 of the XOR-branches only tree.

```
('X',
 [0.75, 0.25],
 [('>',
   ['a_c',
    ('X',
     [0.7142857142857143, 0.2857142857142857],
     ['a_e',
     ('>',
      ['a_f', 'a_b'])])]),
  ('X',
   [0.42857142857142855, 0.5714285714285714],
   ['a_a', 'a_d'])])
```

Figure 32 – Sim tree 2 of the XOR-branches only tree.

10.11.4 Small, combined tree

The three (3) seed combinations are:

Table 66 – Sim tree seeds for the small, combined tree.

	Sim Tree 0	Sim Tree 1	Sim Tree 2
Seed 0	1718538115	990283461	1708917191
Seed 1	1176869713	1969710494	1421704765

The three (3) resulting trees with simulation values are as follows:

```
('+',
 [('X',
   [0.5, 0.5],
   ['a_a',
    ('X',
     [0.4, 0.6],
     ['a_c',
      ('*',
       [0.5357273491284736, 0.3502118621436266, 0.060528983431140826],
       ['a_e',
        ('+',
         ['a_g', 'a_d'])])])]),
  ('+',
   ['a_f', 'a_b'])])
```

Figure 33 – Sim tree 0 of the small, combined tree.

```
('+',
 [('X',
   [0.75, 0.25],
   ['a_a',
    ('X',
     [0.42857142857142855, 0.5714285714285714],
     ['a_c',
      ('*',
       [0.16763358845762388, 0.02554899607507254, 0.0048262906169289415,
        0.002543670576001397,.002102847488469429],
       ['a_e',
        ('+',
         ['a_g', 'a_d'])])])]),
  ('+',
   ['a_f', 'a_b'])])
```

Figure 34 – Sim tree 1 of the small, combined tree.

```
('+',
 [('X',
   [0.5, 0.5],
   ['a_a',
    ('X',
     [0.5555555555555556, 0.4444444444444444],
     ['a_c',
      ('*',
       [0.8266444192850504, 0.6386235175593885, 0.13237504793754443],
       ['a_e',
        ('+',
         ['a_g', 'a_d'])])])]),
  ('+',
   ['a_f', 'a_b'])])
```

Figure 35 – Sim tree 2 of the small, combined tree.

10.11.5 Large, combined tree

The three (3) seed combinations are:

Table 67 – Sim tree seeds for the small, combined tree.

	Sim Tree 0	Sim Tree 1	Sim Tree 2
Seed 0	1602080410	1235104103	1383974871
Seed 1	648465147	925448569	1316528754

The three (3) resulting trees with simulation values are as follows:

```
('X',
 [0.6666666666666666, 0.3333333333333333],
 [('>',
   [('>',
     ['a_c',
      ('*',
       [0.8568863343673961, 0.24253650054053202, 0.22052183610969922],
       ['a_f', 'a_h'])]),
    ('*',
     [0.816236922361096, 0.46883518255370826],
     ['a_a', 'a_i'])]),
  ('+',
   ['a_d',
    ('>',
     ['a_e',
      ('X',
       [0.5, 0.5],
       ['a_b',
        ('*',
         [0.8939180320703698, 0.1273815761114933, 0.009398971481050946,
          0.004211614509921592],
         ['a_g', 'a_j'])])])])])
```

Figure 36 – Sim tree 0 of the large, combined tree.

```
('X',
 [0.14285714285714285, 0.8571428571428571],
 [('>',
   [('>',
     ['a_c',
      ('*',
       [0.5777150804191634, 0.4710819435133378],
       ['a_f', 'a_h'])]),
    ('*',
     [0.9250958647756223, 0.913042964723886, 0.7395763672385849],
     ['a_a', 'a_i'])]),
  ('+',
   ['a_d',
    ('>',
     ['a_e',
      ('X',
       [0.6666666666666666, 0.3333333333333333],
       ['a_b',
        ('*',
         [0.49991190494995685, 0.4581612025276598, 0.3827921113918269],
         ['a_g', 'a_j'])])])])])
```

Figure 37 – Sim tree 1 of the large, combined tree.

```
('X',
 [0.8, 0.2],
 [('>',
   [('>',
     ['a_c',
      ('*',
       [0.8869814147660922, 0.16337736292403074, 0.0316535579857634,
        0.029954469343759848],
       ['a_f', 'a_h'])]),
    ('*',
     [0.4061140759372188, 0.11340518506150618],
     ['a_a', 'a_i'])]),
  ('+',
   ['a_d',
    ('>',
     ['a_e',
      ('X',
       [0.3333333333333333, 0.6666666666666666],
       ['a_b',
        ('*',
         [0.9707367669090233, 0.016479416484125892,
          0.0027768997127055653],
         ['a_g', 'a_j'])])])])])
```

Figure 38 – Sim tree 2 of the large, combined tree

10.12 Process trace generator: Activity sets results

Note that all activities are preceded with an *a_* notation, this is a characteristic of the process trace generator. This applies to all following tables within the **Activity simulation values** section.

10.12.1 Loops only

Activities setup for seed pair *827811881* & *892884050*:

Table 68 – Activities values for sim tree 0.

Activity	Base time value	Base cost value
a_a	389.56	554.08
a_b	605.25	816.05
a_c	922.79	73.60
a_d	935.64	705.25
a_e	428.63	78.12
a_f	573.81	430.15

Activities setup for seed pair *78488783* & *57106468*:

Table 69 – Activities values for sim tree 1.

Activity	Base time value	Base cost value
a_a	448.54	479.72
a_b	293.71	523.92
a_c	580.00	804.40
a_d	678.51	207.63
a_e	506.56	418.31
a_f	391.84	938.46

Activities setup for seed pair *751954824* & *113944411*:

Table 70 – Activities values for sim tree 2.

Activity	Base time value	Base cost value
a_a	243.70	133.20
a_b	963.30	524.01
a_c	311.37	868.37
a_d	928.83	578.36
a_e	358.38	594.22
a_f	510.09	766.15

10.12.2 Parallel branches only

Activities setup for seed pair *1579143339* & *617776679*:

Table 71 – Activities values for sim tree 0.

Activity	Base time value	Base cost value
a_a	620.73	708.82
a_b	267.26	595.22
a_c	269.32	796.28
a_d	717.20	864.05
a_e	497.07	496.11
a_f	674.72	762.85
a_g	711.63	450.31

Activities setup for seed pair *1216797035* & *1143991442*:

Table 72 – Activities values for sim tree 1.

Activity	Base time value	Base cost value
a_a	364.51	531.80
a_b	556.94	317.64
a_c	915.21	157.16

a_d	189.35	49.61
a_e	451.23	815.83
a_f	652.50	368.32
a_g	658.26	113.45

Activities setup for seed pair *1075399334* & *1754616638*:

Table 73 – Activities values for sim tree 2.

Activity	Base time value	Base cost value
a_a	911.73	659.56
a_b	101.31	739.95
a_c	729.33	760.72
a_d	987.40	878.53
a_e	512.90	104.04
a_f	631.32	995.85
a_g	603.97	365.00

10.12.3 XOR-branches only

Activities setup for seed pair *1088179171* & *28661790*:

Table 74 – Activities values for sim tree 0.

Activity	Base time value	Base cost value
a_a	572.93	35.74
a_b	587.60	138.04
a_c	533.66	115.06
a_d	149.41	646.12
a_e	891.79	872.58
a_f	623.84	329.47

Activities setup for seed pair *694198756* & *640368126*:

Table 75 – Activities values for sim tree 1.

Activity	Base time value	Base cost value
a_a	110.39	784.44
a_b	125.32	603.81
a_c	446.68	771.61
a_d	377.64	617.28
a_e	265.62	639.06
a_f	411.48	936.95

Activities setup for seed pair *1150356829* & *1399333154*:

Table 76 – Activities setup for sim tree 2.

Activity	Base time value	Base cost value
a_a	593.02	515.38
a_b	927.14	292.36
a_c	444.58	685.21
a_d	456.21	621.68
a_e	735.63	689.90
a_f	471.68	767.85

10.12.4 Small. combined tree

Activities setup for seed pair *1718538115* & *1176869713*:

Table 77 – Activities values for sim tree 0.

Activity	Base time value	Base cost value
a_a	771.88	977.40
a_b	951.65	188.72
a_c	203.84	714.25

a_d	219.54	75.10
a_e	43.63	354.21
a_f	470.83	976.34
a_g	130.37	976.98

Activities setup for seed pair *990283461* & *1969710494*:

Table 78 – Activities values for sim tree 1.

Activity	Base time value	Base cost value
a_a	97.18	589.34
a_b	453.08	742.08
a_c	835.90	580.68
a_d	267.46	725.02
a_e	250.31	940.56
a_f	128.13	278.27
a_g	165.92	764.08

Activities setup for seed pair *1708917191* & *1421704765*:

Table 79 – Activities values for sim tree 2.

Activity	Base time value	Base cost value
a_a	559.71	713.57
a_b	43.69	83.13
a_c	272.11	758.57
a_d	497.24	782.62
a_e	902.39	276.04
a_f	893.14	513.02
a_g	44.27	483.13

10.12.5 Large. combined tree

Activities setup for seed pair *1602080410* & *64845147*:

Table 80 – Activities values for sim tree 0.

Activity	Base time value	Base cost value
a_a	969.43	117.57
a_b	818.07	642.62
a_c	472.95	279.89
a_d	521.14	448.76
a_e	287.78	826.13
a_f	929.14	91.14
a_g	515.84	762.50
a_h	630.38	156.36
a_i	132.04	677.59
a_j	40.38	577.01

Activities setup for seed pair *1235104103* & *925448569*:

Table 81 – Activities values for sim tree 1.

Activity	Base time value	Base cost value
a_a	565.97	197.66
a_b	987.10	219.09
a_c	189.73	281.01
a_d	660.75	62.35
a_e	949.11	821.85
a_f	650.48	676.52
a_g	840.94	478.25
a_h	585.68	452.00
a_i	684.20	558.07
a_j	581.40	438.18

Activities setup for seed pair *1383974871* & *1316528754*:

Table 82 – Activities values for sim tree 2.

Activity	Base time value	Base cost value
a_a	291.29	364.15
a_b	627.18	286.45
a_c	347.19	518.01
a_d	54.08	803.98
a_e	438.08	881.70
a_f	437.62	808.37
a_g	463.31	116.84
a_h	992.85	331.14
a_i	424.44	262.93
a_j	827.31	14.53

10.13 Process trace generator: Resource sets results

Note that all activities are preceded with an *a_* notation, this is a characteristic of the process trace generator. This applies to all following tables within the **Activity simulation values** section.

10.13.1 Loops only

10.13.1.1 Average resources without random events

Resources setup for seed pair *827811881* & *892884050*:

Table 83 – Average resources without random events seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e	0.920414	-0.000798	0.126148	6	7
r_1	a_b, a_f, a_a, a_e, a_d	0.629925	-0.202731	-0.048200	5	9
r_2	a_f	0.602583	-0.154535	-0.005235	6	5
r_3	a_d, a_e, a_f, a_c, a_b	0.913418	0.044629	-0.042365	6	6
r_4		0.228319	0.414796	0.170198	3	7
r_5		0.777067	0.047427	0.366105	6	5
r_6		0.276988	0.229528	0.347552	3	5
r_7		0.812112	0.408370	0.445150	5	7
r_8		0.689251	0.524925	0.246362	6	9
r_9		0.407127	0.223431	0.151061	5	9

Resources setup for seed pair *78488783* & *571064686*:

Table 84 – Average resources without random events seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_c, a_e, a_a, a_d	0.853426	-0.137848	-0.007110	5	8
r_1	a_a, a_f, a_d, a_e	0.594380	-0.115800	-0.149735	4	7
r_2	a_b, a_c, a_f	0.929502	-0.176845	0.193889	5	8
r_3		0.137582	0.364872	0.093888	4	6
r_4		0.509792	0.486180	0.180184	5	8
r_5		0.359032	0.476135	0.241760	5	8
r_6		0.915971	0.155175	0.213279	4	7
r_7		0.423389	0.285725	0.153991	4	8
r_8		0.642433	0.071598	0.366885	6	7
r_9		0.307391	0.511524	0.141203	3	8

Resources setup for seed pair *751954824* & *113944411*:

Table 85 – Average resources without random events seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e	0.478951	0.153405	-0.077849	3	7
r_1	a_a, a_f	0.941618	0.275939	0.138888	5	6
r_2	a_c, a_f, a_a, a_b, a_d	0.144722	0.074503	0.040048	6	9
r_3	a_b	0.782829	0.281221	-0.091826	5	9
r_4	a_a, a_d	0.749434	-0.018234	-0.131705	6	9

r_5	a_b, a_d, a_e, a_a	0.601922	0.294241	0.187768	3	7
r_6	a_c, a_e, a_f, a_d	0.586414	0.092887	0.065305	5	9
r_7	a_a	0.897172	0.254558	0.156707	5	9
r_8		0.656645	0.057686	0.440117	5	7
r_9		0.262856	0.196418	0.300882	5	6

10.13.1.2 Average resources with random events

The resulting resource parameters for the average resources with random events, is the same as the resulting parameters for the average resources without random events. This is because the addition of a random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.

10.13.1.3 Average resources without random events and with specialized resources

Resources setup for seed pair *827811881* & *892884050*:

Table 86 – Average resources without random events and with specialized resources seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e	0.920414	-0.000798	0.126148	6	7
r_1	a_b, a_f, a_a, a_e, a_d	0.629925	-0.202731	-0.048200	5	9
r_2	a_f	0.602583	-0.154535	-0.005235	6	5
r_3	a_d, a_e, a_f, a_c, a_b	0.913418	0.044629	-0.042365	6	6
r_4		0.228319	0.414796	0.170198	3	7
r_5		0.777067	0.047427	0.366105	6	5
r_6		0.276988	0.229528	0.347552	3	5
r_7		0.812112	0.408370	0.445150	5	7
r_8		0.689251	0.524925	0.246362	6	9
r_9		0.407127	0.223431	0.151061	5	9
Specialized resources						
r_10		0.623892	-0.500000	0.500000		
r_11		0.849139	-0.500000	0.500000		
r_12		0.993314	-0.500000	0.500000		
r_13		0.109843	-0.500000	0.500000		
r_14		0.412405	0.500000	-0.500000		
r_15		0.727870	0.500000	-0.500000		
r_16		0.155966	0.500000	-0.500000		
r_17		0.721926	0.500000	-0.500000		

Resources setup for seed pair *78488783* & *571064686*:

Table 87 – Average resources without random events and with specialized resources seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_c, a_e, a_a, a_d	0.853426	-0.137848	-0.007110	5	8
r_1	a_a, a_f, a_d, a_e	0.594380	-0.115800	-0.149735	4	7
r_2	a_b, a_c, a_f	0.929502	-0.176845	0.193889	5	8
r_3		0.137582	0.364872	0.093888	4	6
r_4		0.509792	0.486180	0.180184	5	8
r_5		0.359032	0.476135	0.241760	5	8

r_6		0.915971	0.155175	0.213279	4	7
r_7		0.423389	0.285725	0.153991	4	8
r_8		0.642433	0.071598	0.366885	6	7
r_9		0.307391	0.511524	0.141203	3	8
Specialized resources						
r_10		0.440189	-0.500000	0.500000		
r_11		0.542991	-0.500000	0.500000		
r_12		0.241690	-0.500000	0.500000		
r_13		0.270567	-0.500000	0.500000		
r_14		0.822326	0.500000	-0.500000		
r_15		0.572093	0.500000	-0.500000		
r_16		0.209232	0.500000	-0.500000		
r_17		0.871395	0.500000	-0.500000		

Resources setup for seed pair *751954824* & *113944411*:

Table 88 – Average resources without random events and with specialized resources seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e	0.478951	0.153405	-0.077849	3	7
r_1	a_a, a_f	0.941618	0.275939	0.138888	5	6
r_2	a_c, a_f, a_a, a_b, a_d	0.144722	0.074503	0.040048	6	9
r_3	a_b	0.782829	0.281221	-0.091826	5	9
r_4	a_a, a_d	0.749434	-0.018234	-0.131705	6	9
r_5	a_b, a_d, a_e, a_a	0.601922	0.294241	0.187768	3	7
r_6	a_c, a_e, a_f, a_d	0.586414	0.092887	0.065305	5	9
r_7	a_a	0.897172	0.254558	0.156707	5	9
r_8		0.656645	0.057686	0.440117	5	7
r_9		0.262856	0.196418	0.300882	5	6
Specialized resources						
r_10		0.221306	-0.500000	0.500000		
r_11		0.176898	-0.500000	0.500000		
r_12		0.268878	-0.500000	0.500000		
r_13		0.375688	-0.500000	0.500000		
r_14		0.385918	0.500000	-0.500000		
r_15		0.691316	0.500000	-0.500000		
r_16		0.913789	0.500000	-0.500000		
r_17		0.366638	0.500000	-0.500000		

10.13.1.4 Average resources with random events and with specialized resources

The resulting resource parameters for the average resources with random events and specialized resources, is the same as the resulting parameters for the average resources without random events and specialized resources. This is because the addition of a random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.

10.13.2 Parallel branches only

10.13.2.1 Average resources without random events

Resources setup for seed pair *1579143339* & *617776679*:

Table 89 – Average resources without random events seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_a, a_d, a_f, a_g, a_e	0.916556	0.264739	0.184737	3	7
r_1	a_f	0.112067	0.007979	-0.149586	5	9
r_2	a_f, a_b, a_a	0.960778	-0.268980	0.057794	4	5
r_3	a_a, a_d, a_c	0.197090	-0.059198	-0.016693	6	6
r_4		0.963751	0.154692	0.090694	6	7
r_5		0.265582	0.000470	0.174146	4	5
r_6		0.875577	0.056296	0.191717	5	5
r_7		0.163531	0.023126	0.187612	3	7
r_8		0.267002	0.163581	0.264897	4	9
r_9		0.272681	0.002322	0.358937	4	9

Resources setup for seed pair *1216797035* & *1143991442*:

Table 90 – Average resources without random events seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_b, a_f, a_e	0.432383	0.130798	0.037686	5	5
r_1	a_f, a_e, a_b, a_c, a_d	0.164676	-0.014706	-0.161370	6	7
r_2	a_c, a_a, a_e, a_b	0.880190	0.169543	0.053301	4	7
r_3		0.377339	0.081320	0.204681	4	5
r_4		0.834130	0.118297	0.360992	4	5
r_5		0.962209	0.453846	0.424382	5	7
r_6		0.535867	0.211120	0.351271	5	7
r_7		0.393349	0.237836	0.447852	5	7
r_8		0.836438	0.151875	0.307891	5	6
r_9		0.109641	0.101172	0.348990	5	7

Resources setup for seed pair *1075399334* & *1754616638*:

Table 91 – Average resources without random events seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_d, a_b, a_e	0.899995	0.020963	0.172626	4	7
r_1	a_f, a_d, a_a, a_g, a_e	0.881519	-0.086564	-0.019295	5	9
r_2	a_a, a_e, a_c, a_b	0.158848	-0.064966	-0.171615	5	8
r_3	a_a, a_e, a_g, a_c, a_f	0.268773	0.108624	0.047091	4	7
r_4	a_a, a_f, a_e	0.114809	0.148080	0.175293	6	9
r_5	a_d, a_c, a_a	0.349823	0.252586	-0.090544	3	6
r_6	a_a, a_g, a_e, a_b, a_d, a_c	0.555626	0.142905	-0.060893	3	6
r_7	a_a, a_f, a_d, a_c, a_b, a_g	0.756382	-0.273644	-0.163232	6	6
r_8		0.699128	0.499312	0.410396	4	9
r_9		0.605663	0.270403	0.356160	4	9

10.13.2.2 Average resources with random events

The resulting resource parameters for the average resources with random events, is the same as the resulting parameters for the average resources without random events. This is because the addition of a

random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.

10.13.2.3 Average resources without random events and with specialized resources

Resources setup for seed pair *1579143339* & *617776679*:

Table 92 – Average resources without random events and with specialized resources seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_a, a_d, a_f, a_g, a_e	0.916556	0.264739	0.184737	3	7
r_1	a_f	0.112067	0.007979	-0.149586	5	9
r_2	a_f, a_b, a_a	0.960778	-0.268980	0.057794	4	5
r_3	a_a, a_d, a_c	0.197090	-0.059198	-0.016693	6	6
r_4		0.963751	0.154692	0.090694	6	7
r_5		0.265582	0.000470	0.174146	4	5
r_6		0.875577	0.056296	0.191717	5	5
r_7		0.163531	0.023126	0.187612	3	7
r_8		0.267002	0.163581	0.264897	4	9
r_9		0.272681	0.002322	0.358937	4	9
Specialized resources						
r_10		0.809576	-0.500000	0.500000		
r_11		0.453388	-0.500000	0.500000		
r_12		0.180483	-0.500000	0.500000		
r_13		0.589070	-0.500000	0.500000		
r_14		0.250666	0.500000	-0.500000		
r_15		0.291038	0.500000	-0.500000		
r_16		0.786255	0.500000	-0.500000		
r_17		0.878807	0.500000	-0.500000		

Resources setup for seed pair *1216797035* & *1143991442*:

Table 93 – Average resources without random events and with specialized resources seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_b, a_f, a_e	0.432383	0.130798	0.037686	5	5
r_1	a_f, a_e, a_b, a_c, a_d	0.164676	-0.014706	-0.161370	6	7
r_2	a_c, a_a, a_e, a_b	0.880190	0.169543	0.053301	4	7
r_3		0.377339	0.081320	0.204681	4	5
r_4		0.834130	0.118297	0.360992	4	5
r_5		0.962209	0.453846	0.424382	5	7
r_6		0.535867	0.211120	0.351271	5	7
r_7		0.393349	0.237836	0.447852	5	7
r_8		0.836438	0.151875	0.307891	5	6
r_9		0.109641	0.101172	0.348990	5	7
Specialized resources						
r_10		0.484974	-0.500000	0.500000		
r_11		0.886981	-0.500000	0.500000		
r_12		0.223994	-0.500000	0.500000		
r_13		0.826126	-0.500000	0.500000		
r_14		0.959755	0.500000	-0.500000		
r_15		0.276022	0.500000	-0.500000		

r_16		0.877033	0.500000	-0.500000		
r_17		0.722597	0.500000	-0.500000		

Resources setup for seed pair *1075399334* & *1754616638*:

Table 94 – Average resources without random events and with specialized resources seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_d, a_b, a_e	0.899995	0.020963	0.172626	4	7
r_1	a_f, a_d, a_a, a_g, a_e	0.881519	-0.086564	-0.019295	5	9
r_2	a_a, a_e, a_c, a_b	0.158848	-0.064966	-0.171615	5	8
r_3	a_a, a_e, a_g, a_c, a_f	0.268773	0.108624	0.047091	4	7
r_4	a_a, a_f, a_e	0.114809	0.148080	0.175293	6	9
r_5	a_d, a_c, a_a	0.349823	0.252586	-0.090544	3	6
r_6	a_a, a_g, a_e, a_b, a_d, a_c	0.555626	0.142905	-0.060893	3	6
r_7	a_a, a_f, a_d, a_c, a_b, a_g	0.756382	-0.273644	-0.163232	6	6
r_8		0.699128	0.499312	0.410396	4	9
r_9		0.605663	0.270403	0.356160	4	9
Specialized resources						
r_10		0.519840	-0.500000	0.500000		
r_11		0.125580	-0.500000	0.500000		
r_12		0.267946	-0.500000	0.500000		
r_13		0.414269	-0.500000	0.500000		
r_14		0.898345	0.500000	-0.500000		
r_15		0.519577	0.500000	-0.500000		
r_16		0.125831	0.500000	-0.500000		
r_17		0.826258	0.500000	-0.500000		

10.13.2.4 Average resources with random events and with specialized resources

The resulting resource parameters for the average resources with random events and specialized resources, is the same as the resulting parameters for the average resources without random events and specialized resources. This is because the addition of a random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.

10.13.3 XOR-branches only

10.13.3.1 Average resources without random events

Resources setup for seed pair *1088179171* & *28661790*:

Table 95 – Average resources without random events seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_c, a_f, a_a, a_d, a_b	0.157766	0.224964	0.048459	3	9
r_1	a_e, a_a, a_c, a_d	0.121389	-0.100670	-0.176506	5	5
r_2	a_f, a_d	0.984564	-0.069585	-0.095482	5	5
r_3	a_d, a_a, a_b	0.183523	-0.286176	-0.192829	5	5
r_4	a_b	0.958239	0.007910	-0.134083	5	9
r_5		0.408517	0.284779	0.090525	3	5

r_6		0.458328	-0.009812	0.286588	5	5
r_7		0.129026	0.154388	0.160531	5	5
r_8		0.528588	0.400971	0.216075	5	5
r_9		0.924357	0.286597	0.355690	4	9

Resources setup for seed pair *694198756* & *640368126*:

Table 96 – Average resources without random events seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e, a_c, a_d, a_a, a_f	0.665650	0.225169	-0.162403	4	5
r_1	a_b	0.464529	-0.210180	0.155700	6	7
r_2	a_a	0.465674	0.172777	-0.125100	4	8
r_3	a_b	0.404228	-0.196395	-0.110553	3	8
r_4	a_d, a_b	0.264668	-0.130066	-0.028440	4	8
r_5	a_d, a_a, a_e	0.831119	0.160920	0.029610	4	8
r_6	a_c, a_b, a_a	0.118725	0.266444	-0.046341	5	8
r_7		0.353122	0.254724	0.418194	6	8
r_8		0.190952	0.313599	0.130550	5	8
r_9		0.788614	0.500437	0.195802	5	7

Resources setup for seed pair *1150356829* & *1399333154*:

Table 97 – Average resources without random events seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e, a_f, a_c	0.642402	-0.187205	0.166488	6	9
r_1	a_f	0.502514	-0.080209	-0.161191	6	5
r_2	a_f, a_a, a_e	0.195869	0.199047	0.054236	6	6
r_3		0.200671	0.033682	0.398578	3	7
r_4		0.892344	0.390513	0.222255	5	9
r_5		0.630866	0.246190	0.396772	6	5
r_6		0.837432	0.051199	0.402704	4	6
r_7		0.888102	0.088863	0.217978	5	9
r_8		0.849076	0.154073	0.434445	3	9
r_9		0.238601	-0.019835	0.385108	5	5

10.13.3.2 Average resources with random events

The resulting resource parameters for the average resources with random events, is the same as the resulting parameters for the average resources without random events. This is because the addition of a random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.

10.13.3.3 Average resources without random events and with specialized resources

Resources setup for seed pair *1088179171* & *28661790*:

Table 98 – Average resources without random events and with specialized resources seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_c, a_f, a_a, a_d, a_b	0.157766	0.224964	0.048459	3	9
r_1	a_e, a_a, a_c, a_d	0.121389	-0.100670	-0.176506	5	5
r_2	a_f, a_d	0.984564	-0.069585	-0.095482	5	5

r_3	a_d, a_a, a_b	0.183523	-0.286176	-0.192829	5	5
r_4	a_b	0.958239	0.007910	-0.134083	5	9
r_5		0.408517	0.284779	0.090525	3	5
r_6		0.458328	-0.009812	0.286588	5	5
r_7		0.129026	0.154388	0.160531	5	5
r_8		0.528588	0.400971	0.216075	5	5
r_9		0.924357	0.286597	0.355690	4	9
Specialized resources						
r_10		0.470618	-0.500000	0.500000		
r_11		0.357029	-0.500000	0.500000		
r_12		0.625171	-0.500000	0.500000		
r_13		0.891407	-0.500000	0.500000		
r_14		0.176860	0.500000	-0.500000		
r_15		0.759846	0.500000	-0.500000		
r_16		0.526806	0.500000	-0.500000		
r_17		0.721455	0.500000	-0.500000		

Resources setup for seed pair *694198756* & *640368126*:

Table 99 – Average resources without random events and with specialized resources seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e, a_c, a_d, a_a, a_f	0.665650	0.225169	-0.162403	4	5
r_1	a_b	0.464529	-0.210180	0.155700	6	7
r_2	a_a	0.465674	0.172777	-0.125100	4	8
r_3	a_b	0.404228	-0.196395	-0.110553	3	8
r_4	a_d, a_b	0.264668	-0.130066	-0.028440	4	8
r_5	a_d, a_a, a_e	0.831119	0.160920	0.029610	4	8
r_6	a_c, a_b, a_a	0.118725	0.266444	-0.046341	5	8
r_7		0.353122	0.254724	0.418194	6	8
r_8		0.190952	0.313599	0.130550	5	8
r_9		0.788614	0.500437	0.195802	5	7
Specialized resources						
r_10		0.454024	-0.500000	0.500000		
r_11		0.777729	-0.500000	0.500000		
r_12		0.241361	-0.500000	0.500000		
r_13		0.275182	-0.500000	0.500000		
r_14		0.938151	0.500000	-0.500000		
r_15		0.884341	0.500000	-0.500000		
r_16		0.911585	0.500000	-0.500000		
r_17		0.841864	0.500000	-0.500000		

Resources setup for seed pair *1150356829* & *1399333154*:

Table 100 – Average resources without random events and with specialized resources seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e, a_f, a_c	0.642402	-0.187205	0.166488	6	9
r_1	a_f	0.502514	-0.080209	-0.161191	6	5
r_2	a_f, a_a, a_e	0.195869	0.199047	0.054236	6	6
r_3		0.200671	0.033682	0.398578	3	7
r_4		0.892344	0.390513	0.222255	5	9
r_5		0.630866	0.246190	0.396772	6	5
r_6		0.837432	0.051199	0.402704	4	6
r_7		0.888102	0.088863	0.217978	5	9
r_8		0.849076	0.154073	0.434445	3	9

r_9		0.238601	-0.019835	0.385108	5	5
Specialized resources						
r_10		0.989276	-0.500000	0.500000		
r_11		0.726084	-0.500000	0.500000		
r_12		0.685680	-0.500000	0.500000		
r_13		0.311557	-0.500000	0.500000		
r_14		0.757203	0.500000	-0.500000		
r_15		0.808933	0.500000	-0.500000		
r_16		0.243324	0.500000	-0.500000		
r_17		0.274912	0.500000	-0.500000		

10.13.3.4 Average resources with random events and with specialized resources

The resulting resource parameters for the average resources with random events and specialized resources, is the same as the resulting parameters for the average resources without random events and specialized resources. This is because the addition of a random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.

10.13.4 Small, combined tree

10.13.4.1 Average resources without random events

Resources setup for seed pair *1718538115 & 1176869713*:

Table 101 – Average resources without random events seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_a	0.158015	-0.010268	0.132464	3	7
r_1	a_g	0.781009	0.216598	0.042697	4	7
r_2	a_e, a_c, a_f, a_g, a_d	0.236172	-0.063923	0.086787	4	6
r_3	a_f	0.832822	0.216360	0.107416	3	7
r_4	a_g, a_e, a_d	0.794436	-0.292807	0.156056	3	6
r_5	a_d, a_c, a_e, a_a	0.129981	0.297535	0.114275	4	8
r_6	a_e	0.949316	0.167332	0.188116	4	8
r_7	a_d, a_a, a_f	0.157495	-0.242720	0.077790	6	7
r_8		0.501517	0.012107	0.393627	4	8
r_9		0.802496	0.390332	0.345039	3	6

Resources setup for seed pair *990283461 & 1969710494*:

Table 102 – Average resources without random events seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_e, a_g, a_b, a_d	0.536559	-0.009791	-0.039098	4	9
r_1	a_e, a_d	0.815651	0.145446	0.143597	6	9
r_2		0.496044	0.222588	0.239230	4	6
r_3		0.507935	0.251795	0.127812	4	8
r_4		0.615895	0.234606	0.233457	4	5
r_5		0.131972	0.086995	0.313685	6	5
r_6		0.521752	0.546660	0.304834	4	6
r_7		0.929053	0.043002	0.301377	4	8
r_8		0.287014	0.342198	0.074367	6	8
r_9		0.244082	0.151777	0.266208	6	5

Resources setup for seed pair *1708917191 & 1421704765*:

Table 103 – Average resources without random events seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_a, a_g	0.717161	0.267873	0.198809	6	7
r_1	a_d, a_a	0.181336	0.090986	-0.023921	5	7
r_2	a_c	0.712825	0.058402	-0.101056	6	6
r_3	a_f, a_a, a_c	0.196962	-0.296524	-0.193763	6	5
r_4		0.231194	0.466476	0.239329	6	5
r_5		0.663873	0.043717	0.060791	4	6
r_6		0.233476	0.337214	0.154498	3	6
r_7		0.540605	0.125325	0.443224	6	7
r_8		0.838832	0.172906	0.086659	4	7
r_9		0.323515	0.254124	0.343637	6	7

10.13.4.2 Average resources with random events

The resulting resource parameters for the average resources with random events, is the same as the resulting parameters for the average resources without random events. This is because the addition of a random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.

10.13.4.3 Average resources without random events and with specialized resources

Resources setup for seed pair *1718538115* & *1176869713*:

Table 104 – Average resources without random events and with specialized resources seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_a	0.158015	-0.010268	0.132464	3	7
r_1	a_g	0.781009	0.216598	0.042697	4	7
r_2	a_e, a_c, a_f, a_g, a_d	0.236172	-0.063923	0.086787	4	6
r_3	a_f	0.832822	0.216360	0.107416	3	7
r_4	a_g, a_e, a_d	0.794436	-0.292807	0.156056	3	6
r_5	a_d, a_c, a_e, a_a	0.129981	0.297535	0.114275	4	8
r_6	a_e	0.949316	0.167332	0.188116	4	8
r_7	a_d, a_a, a_f	0.157495	-0.242720	0.077790	6	7
r_8		0.501517	0.012107	0.393627	4	8
r_9		0.802496	0.390332	0.345039	3	6
Specialized resources						
r_10		0.632957	-0.500000	0.500000		
r_11		0.885364	-0.500000	0.500000		
r_12		0.349910	-0.500000	0.500000		
r_13		0.302426	-0.500000	0.500000		
r_14		0.714547	0.500000	-0.500000		
r_15		0.834851	0.500000	-0.500000		
r_16		0.399971	0.500000	-0.500000		
r_17		0.919343	0.500000	-0.500000		

Resources setup for seed pair *990283461* & *1969710494*:

Table 105 – Average resources without random events and with specialized resources seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)

r_0	a_e, a_g, a_b, a_d	0.536559	-0.009791	-0.039098	4	9
r_1	a_e, a_d	0.815651	0.145446	0.143597	6	9
r_2		0.496044	0.222588	0.239230	4	6
r_3		0.507935	0.251795	0.127812	4	8
r_4		0.615895	0.234606	0.233457	4	5
r_5		0.131972	0.086995	0.313685	6	5
r_6		0.521752	0.546660	0.304834	4	6
r_7		0.929053	0.043002	0.301377	4	8
r_8		0.287014	0.342198	0.074367	6	8
r_9		0.244082	0.151777	0.266208	6	5
Specialized resources						
r_10		0.905114	-0.500000	0.500000		
r_11		0.101776	-0.500000	0.500000		
r_12		0.941368	-0.500000	0.500000		
r_13		0.149639	-0.500000	0.500000		
r_14		0.885854	0.500000	-0.500000		
r_15		0.527485	0.500000	-0.500000		
r_16		0.666293	0.500000	-0.500000		
r_17		0.594059	0.500000	-0.500000		

Resources setup for seed pair *1708917191* & *1421704765*:

Table 106 – Average resources without random events and with specialized resources seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	[a_a, a_g]	0.717161	0.267873	0.198809	6	7
r_1	[a_d, a_a]	0.181336	0.090986	-0.023921	5	7
r_2	[a_c]	0.712825	0.058402	-0.101056	6	6
r_3	[a_f, a_a, a_c]	0.196962	-0.296524	-0.193763	6	5
r_4		0.231194	0.466476	0.239329	6	5
r_5		0.663873	0.043717	0.060791	4	6
r_6		0.233476	0.337214	0.154498	3	6
r_7		0.540605	0.125325	0.443224	6	7
r_8		0.838832	0.172906	0.086659	4	7
r_9		0.323515	0.254124	0.343637	6	7
Specialized resources						
r_10		0.381256	-0.500000	0.500000		
r_11		0.737904	-0.500000	0.500000		
r_12		0.309329	-0.500000	0.500000		
r_13		0.493560	-0.500000	0.500000		
r_14		0.567350	0.500000	-0.500000		
r_15		0.550823	0.500000	-0.500000		
r_16		0.764538	0.500000	-0.500000		
r_17		0.712569	0.500000	-0.500000		

10.13.4.4 Average resources with random events and with specialized resources

The resulting resource parameters for the average resources with random events and specialized resources, is the same as the resulting parameters for the average resources without random events and specialized resources. This is because the addition of a random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.

10.13.5 Large, combined tree

10.13.5.1 Average resources without random events

Resources setup for seed pair *1602080410* & *64845147*:

Table 107 – Average resources without random events seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_h, a_f, a_c, a_a	0.825431	0.250171	0.135195	6	6
r_1		0.523014	0.083562	0.448967	4	6
r_2		0.155295	0.277025	0.209048	5	6
r_3		0.937392	0.071287	0.169014	5	8
r_4		0.553726	0.537893	0.172952	5	8
r_5		0.586393	0.485045	0.376728	4	8
r_6		0.453205	0.084579	0.360978	4	8
r_7		0.895183	0.225772	0.362027	3	7
r_8		0.492710	-0.026345	0.058214	4	6
r_9		0.313917	0.282066	0.293429	3	7

Resources setup for seed pair *1235104103* & *925448569*:

Table 108 – Average resources without random events seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_j, a_a, a_g, a_c, a_d, a_h, a_e, a_b	0.179561	-0.052460	-0.055994	3	6
r_1	a_c	0.958650	-0.035791	0.075692	3	8
r_2	a_a, a_d	0.930285	-0.146778	-0.091530	5	5
r_3	a_h, a_i, a_b, a_c, a_e, a_g	0.373655	0.261319	0.188411	6	7
r_4	a_a, a_e, a_i, a_b, a_h, a_f, a_g	0.136513	0.107790	-0.153667	3	8
r_5	a_j, a_c, a_i, a_g, a_e, a_f, a_d	0.660809	0.067160	-0.166120	3	5
r_6		0.471310	0.170879	0.104655	3	7
r_7		0.445688	0.423160	0.132171	5	5
r_8		0.904943	0.463863	0.311865	4	7
r_9		0.758423	0.333638	0.322176	6	7

Resources setup for seed pair *1383974871* & *1316528754*:

Table 109 – Average resources without random events seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_g, a_b, a_f, a_h, a_i, a_c, a_a, a_d, a_j	0.652956	-0.037458	-0.094767	3	5
r_1	a_j, a_d, a_h, a_f, a_i, a_g, a_b, a_c, a_e	0.259693	0.088016	-0.149701	5	8
r_2	a_h, a_g, a_e	0.412760	0.057443	-0.017310	4	8
r_3	a_g	0.505690	0.127552	0.182860	3	8
r_4	a_f, a_e, a_b, a_i, a_a, a_d	0.585027	-0.294220	0.047259	3	5
r_5	a_b, a_i, a_g, a_h, a_f	0.758280	0.247651	-0.182290	4	8
r_6	a_f, a_a	0.178666	-0.282778	0.085961	6	5

r_7	a_f, a_c, a_i, a_b, a_e, a_j	0.238206	-0.109496	0.146479	5	8
r_8		0.537962	0.179948	0.166671	5	5
r_9		0.565356	0.089500	0.083838	5	5

10.13.5.2 Average resources with random events

The resulting resource parameters for the average resources with random events, is the same as the resulting parameters for the average resources without random events. This is because the addition of a random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.

10.13.5.3 Average resources without random events and with specialized resources

Resources setup for seed pair *1602080410* & *64845147*:

Table 110 – Average resources without random events and with specialized resources seeds from sim tree 0.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_h, a_f, a_c, a_a	0.825431	0.250171	0.135195	6	6
r_1		0.523014	0.083562	0.448967	4	6
r_2		0.155295	0.277025	0.209048	5	6
r_3		0.937392	0.071287	0.169014	5	8
r_4		0.553726	0.537893	0.172952	5	8
r_5		0.586393	0.485045	0.376728	4	8
r_6		0.453205	0.084579	0.360978	4	8
r_7		0.895183	0.225772	0.362027	3	7
r_8		0.492710	-0.026345	0.058214	4	6
r_9		0.313917	0.282066	0.293429	3	7
Specialized resources						
r_10		0.265972	-0.500000	0.500000		
r_11		0.684984	-0.500000	0.500000		
r_12		0.627025	-0.500000	0.500000		
r_13		0.425247	-0.500000	0.500000		
r_14		0.147976	0.500000	-0.500000		
r_15		0.742781	0.500000	-0.500000		
r_16		0.663366	0.500000	-0.500000		
r_17		0.403060	0.500000	-0.500000		

Resources setup for seed pair *1235104103* & *925448569*:

Table 111 – Average resources without random events and with specialized resources seeds from sim tree 1.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_j, a_a, a_g, a_c, a_d, a_h, a_e, a_b	0.179561	-0.052460	-0.055994	3	6
r_1	a_c	0.958650	-0.035791	0.075692	3	8
r_2	a_a, a_d	0.930285	-0.146778	-0.091530	5	5
r_3	a_h, a_i, a_b, a_c, a_e, a_g	0.373655	0.261319	0.188411	6	7
r_4	a_a, a_e, a_i, a_b, a_h, a_f, a_g	0.136513	0.107790	-0.153667	3	8

r_5	a_j, a_c, a_i, a_g, a_e, a_f, a_d	0.660809	0.067160	-0.166120	3	5
r_6		0.471310	0.170879	0.104655	3	7
r_7		0.445688	0.423160	0.132171	5	5
r_8		0.904943	0.463863	0.311865	4	7
r_9		0.758423	0.333638	0.322176	6	7
Specialized resources						
r_10		0.583049	-0.500000	0.500000		
r_11		0.503620	-0.500000	0.500000		
r_12		0.266339	-0.500000	0.500000		
r_13		0.281288	-0.500000	0.500000		
r_14		0.680562	0.500000	-0.500000		
r_15		0.282343	0.500000	-0.500000		
r_16		0.342686	0.500000	-0.500000		
r_17		0.836051	0.500000	-0.500000		

Resources setup for seed pair *1383974871 & 1316528754*:

Table 112 – Average resources without random events and with specialized resources seeds from sim tree 2.

Resource	Limited to activities	Allocation probability	Time modifier	Cost modifier	Threshold time function (τ)	Threshold cost function (τ)
r_0	a_g, a_b, a_f, a_h, a_i, a_c, a_a, a_d, a_j	0.652956	-0.037458	-0.094767	3	5
r_1	a_j, a_d, a_h, a_f, a_i, a_g, a_b, a_c, a_e	0.259693	0.088016	-0.149701	5	8
r_2	a_h, a_g, a_e	0.412760	0.057443	-0.017310	4	8
r_3	a_g	0.505690	0.127552	0.182860	3	8
r_4	a_f, a_e, a_b, a_i, a_a, a_d	0.585027	-0.294220	0.047259	3	5
r_5	a_b, a_i, a_g, a_h, a_f	0.758280	0.247651	-0.182290	4	8
r_6	a_f, a_a	0.178666	-0.282778	0.085961	6	5
r_7	a_f, a_c, a_i, a_b, a_e, a_j	0.238206	-0.109496	0.146479	5	8
r_8		0.537962	0.179948	0.166671	5	5
r_9		0.565356	0.089500	0.083838	5	5
Specialized resources						
r_10		0.920015	-0.500000	0.500000		
r_11		0.968996	-0.500000	0.500000		
r_12		0.846316	-0.500000	0.500000		
r_13		0.863480	-0.500000	0.500000		
r_14		0.683099	0.500000	-0.500000		
r_15		0.550267	0.500000	-0.500000		
r_16		0.821394	0.500000	-0.500000		
r_17		0.577061	0.500000	-0.500000		

10.13.5.4 Average resources with random events and with specialized resources

The resulting resource parameters for the average resources with random events and specialized resources, is the same as the resulting parameters for the average resources without random events and specialized resources. This is because the addition of a random function does not impact the random generation of the "constant" parameters as shown in the tables.

Please refer to the setup above for the parameter settings of this setup.