

UNIVERSITY OF TARTU  
Faculty of Science and Technology  
Institute of Computer Science  
Software Engineering Curriculum

Jonas Berx

# Optimisation of Business Processes with Differentiated Resources

Master's Thesis (30 ECTS)

Supervisor(s): Orlenys López-Pintado, PhD  
Marlon Dumas, PhD

Tartu 2023

# Business Process Optimisation of Differentiated Resources

## **Abstract:**

Business process optimisation focuses on improving one or more performance measures in a business process model. Improving a business process can be achieved by making changes to the resource allocations within that process. Resource allocation is the distribution of available resources in a process to fulfill the organisation goals. The misconfiguration of resource allocations can heavily impact the cost and cycle time of the process. To address this issue, optimisation approaches have to be adapted to include differentiated resources. Differentiated resources each have their calendar, allowing for better customisability of the resource allocation and in turn better optimisation of the process model. This thesis proposes a multi-objective resource allocation optimisation approach for business processes with differentiated resources to minimise the cost and cycle time. The approach heuristically searches the space of possible resource allocations using simulation models to evaluate each resource allocation. An empirical evaluation shows that iteratively optimising resource allocations in conjunction with resource calendars lead to superior cost-time tradeoffs for optimising these allocations and calendars separately. Additionally, this thesis implements a web application to facilitate the interaction experience with the proposed optimisation algorithm.

## **Keywords:**

Business process optimisation, Roster optimisation, Resource allocation, Multi-objective optimisation, Process simulation

**CERCS:** P170 - Computer science, numerical analysis, systems, control

## Äriprotsesside optimeerimine diferentseeritud ressurssidega

### Lühikokkuvõte:

Äriprotsesside optimeerimine keskendub äriprotsessimudeli ühe või mitme jõudlusnäitaja täiustamisele. Äriprotsessi saab täiustada, tehes selle protsessi raames ressursside eraldamises muudatusi. Ressursi jaotamine on tavaliselt protsessi kulude ja tsükliaja vaheline kompromiss. Peavoolu protsesside optimeerimisel on mitmesuguseid piiranguid, millest kõige silmatorkavam piirang tuleneb asjaolust, et nad käsitlevad ressursse diferentseerimata üksustena, mis on rühmitatud kogumitesse. Varasemad uuringud on neid eeldusi tunnistanud, ilma et oleks kvantifitseeritud nende mõju simulatsioonimudeli täpsusele ja omakorda optimeerimise täpsusele. Seejärel eeldatakse, et kõik need basseinis olevad ressursid on sama jõudlusega ja jagavad sama kalendrit. Diferentseerimata ressursside vastand on diferentseeritud ressursid. Igal diferentseeritud ressursil on oma kalender, mis võimaldab ressursside jaotamist paremini kohandada ja protsessimudelit omakorda paremini optimeerida. See lõputöö pakub välja mitme eesmärgiga ressursside jaotamise optimeerimise lähenemisviisi äriprotsesside jaoks, millel on kulude ja tsükliaja osas diferentseeritud ressurss. Lähenemisviis otsib valikuliselt läbi võimalike nimekirjade eraldamise ruumi, kasutades simulatsioonimudelit, et hinnata iga nimekirja jaotust. Empiiriline hindamine näitab, et ressursside jaotamise iteratiivne optimeerimine koos ressursikalendritega annab paremaid kulu-aja kompromisse nende jaotuste ja kalendrite eraldi optimeerimisel.

### Võtmesõnad:

Äriprotsesside optimeerimine, nimekirja optimeerimine, ressursside eraldamine, mitme eesmärgi optimeerimine, protsessi simulatsioon

**CERCS:** P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria).

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>6</b>  |
| <b>2</b> | <b>Background and related work</b>  | <b>10</b> |
| 2.1      | Business Process Model . . . . .  | 10        |
| 2.2      | Simulation Model with Differentiated Resources . . . . .                            | 11        |
| 2.3      | Pareto Fronts and Meta-Heuristic Optimisation Algorithms . .                        | 12        |
| 2.4      | Resource optimisation in business processes . . . . .                               | 13        |
| <b>3</b> | <b>Design and Architecture</b>  | <b>16</b> |
| 3.1      | Resource Allocation and Roster Optimisation with Differentiated Resources . . . . . | 16        |
| 3.1.1    | Pareto Front and Objective Functions . . . . .                                      | 17        |
| 3.1.2    | Hill-Climbing Meta-heuristic . . . . .  | 18        |
| 3.2      | Architecture and Deployment . . . . .   | 24        |
| 3.2.1    | Architecture Overview . . . . .   | 24        |
| 3.2.2    | Deployment . . . . .  | 28        |
| 3.2.3    | Architecture Summary . . . . .  | 29        |
| <b>4</b> | <b>User Interface</b>   | <b>30</b> |
| 4.1      | UI Development . . . . .  | 30        |
| 4.2      | UI Walkthrough . . . . .  | 30        |
| 4.2.1    | Landing page . . . . .  | 31        |
| 4.2.2    | Global Constraints page . . . . .   | 31        |
| 4.2.3    | Scenario Constraints page . . . . .   | 33        |
| 4.2.4    | Resource Constraints page . . . . .   | 34        |
| 4.2.5    | Simulation Results page . . . . .   | 38        |
| <b>5</b> | <b>Implementation and Evaluation</b>  | <b>40</b> |
| 5.1      | Datasets . . . . .  | 40        |
| 5.2      | Experimental setup . . . . .  | 42        |
| 5.3      | Experimental Results . . . . .  | 44        |
| <b>6</b> | <b>Conclusion and future work</b>   | <b>49</b> |
|          | <b>Acknowledgement</b>  | <b>50</b> |
|          | <b>References</b>   | <b>52</b> |

|  |           |
|--|-----------|
| <b>Appendix</b>  | <b>53</b> |
| I.I Dockerfile - presentation tier . . . . .                   | 53        |
| I.II Docker-compose configuration - application tier . . . . . | 54        |
| II. Licence . . . . .  | 55        |

# 1. Introduction

Business Process Optimisation (**BPO**) [1] is a technique that focuses on improving a business's efficiency by optimising its processes in a targeted manner. The starting point for BPO is a snapshot of the current performance of the business process. To make this snapshot, you need the simulation model, consisting of a process model, enhanced with the parameters to capture the model's performance, e.g., the available resources and activity processing times. When organisations wish to improve their business process, they speak about the cycle time, the total amount of time it takes to complete one instance of the process, and/or the cost, the total cost it takes to complete one instance of the process.

The issue of resource allocation of differentiated resources involves determining the optimal amount of resources to assign to each activity in a process to either maximise or minimise one or more performance measures. When a process has a higher number of resources, the resources are less busy and have lower utilisation. Contrary, when a process has fewer resources allocated, the resources are busier and have higher utilisation. Although higher resource utilisation leads to a lower cost per instance, it also results in longer waiting times due to resource contention. Conversely, lower utilisation leads to a higher cost per instance but shorter waiting times. The problem is finding a balance between minimising costs and reducing waiting times. There is no single solution that can minimise both cost and time simultaneously. Instead, there are a set of optimal solutions, known as the Pareto Front, where no objective, such as time and cost can be improved without sacrificing another.

The resources that perform in the business process greatly influence the cycle time and cost. An incorrect configuration of resources results in a higher cycle time and cost of the process and is thus undesired. E.g., In a loan application process, there are two resources: the Junior Loan Officer (*JLO*) and the Senior Loan Officer (*SLO*). The (*SLO*) is also dependent on completing a task assigned to the (*JLO*). The (*JLO*)'s tasks arrive in the morning, but they are only assigned to work in the afternoon. This misconfiguration leads to waiting times because the resource is unavailable when the task arrives. Furthermore, because the (*JLO*) cannot complete their task in time, the (*SLO*) cannot continue with their work as they are relying on the (*JLO*) to finish first. Sometimes, a misconfiguration may not impact the cycle time and cost a lot. However, many misconfigured resources can inflate this cycle time and cost heavily. E.g., in an administrative process

with Clerks, with an increased workload between 12.00 and 16.00. To manage the workload between those times, at least fifteen Clerks should be assigned. In reality, only five Clerks are available during those times and ten Clerks work during less busy hours. As a result, a backlog of unfinished work starts building up and cases take longer to complete; the waiting time increases. In turn, the cost per instance also increases because the resources are still being paid, even though they may not be working; the cost per instance increases.

Given that in classic, undifferentiated BP simulation models, resource pools are disjoint; they cannot capture scenarios where participants share their time across multiple resource pools. Also, since all resources in a resource pool have the same timetable, these models cannot capture scenarios where a pool incorporates some part-time and full-time resources. Secondly, in classic, undifferentiated BP simulation models, the processing times of an activity do not depend on the resource that performs it. Hence, such models cannot capture scenarios where some resources in a resource pool are faster or slower than others.

This thesis will consider resources from now on as differentiated resources that are more customisable than the current business process simulation and optimisation standard with undifferentiated resources. To achieve our goal, we must acknowledge the following observations:

- **O1** *Unpooled resource allocation.* Resources do not belong to a resource pool, a.k.a. a collection of resources that are assigned to perform specific tasks within a business process. However, resources can perform the same tasks or a subset of tasks.
- **O2** *Differentiated performance* The processing time of an activity, the time it takes for the activity to be completed from start to finish, is dependent on the resource that performs it.
- **O3** *Differentiated availability* Each resource in the process has their own calendar, e.g., Accountant1 and Accountant2 perform the same tasks but are not necessarily working during the same time.

We propose an approach to optimise resource calendars to address the cost and time inefficiencies due to the misallocation of resources. For example, a task with high waiting times can indicate that the availability calendar of a resource is not aligned with the arrival times of its assigned tasks in a process, i.e., the resources are not available when required. Accordingly, the

thesis advocates a perturbation approach that extends (where possible) the calendars of underutilised resources (or removes resources altogether) while shrinking the calendar of over-utilised resources (or adding new resources). These operations alter the number of resources available at different periods in time, i.e. the *resource roster*. Given these possible perturbations, this research proposes an approach to optimise the roster of resources in a business process with respect to resource cost and cycle time under the assumption of differentiated resource calendars. We wish to answer the following research question:

- **RQ1** - How can we simultaneously improve the cycle time and cost of a business process with differentiated resources?

We will try to answer this question by retrieving a set of optimal solutions that simultaneously minimise both cycle time and cost, following a multi-objective optimisation approach. We decided to follow the **Design Science** methodology because we aim to solve a problem by developing and evaluating a software artefact. Its guidelines state to identify the problem first and motivate its solution. The second step is to define the objectives of the solution and, afterwards, to design and develop the solution. Finally, the developed artefact should be demonstrated and evaluated. The results are then communicated, and possible improvements are discussed. This process can be repeated iteratively to produce a better solution artefact in each iteration. The steps of the methodology framework used in this thesis are presented in Figure 1.

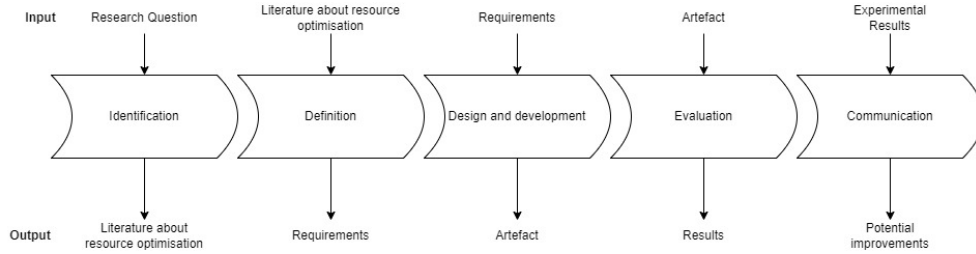


Figure 1: Research methodology approach

During the *identification step*, the problem that needs to be resolved is identified. This step identifies the gap in the literature and poses the research question. We also filter out less relevant literature using knock-out checks during this step. In the *definition step*, the focus is on specifying the criteria



that the solution should fulfil. The discovered gap in the literature calls for a more accurate optimisation approach capable of improving process models with differentiated resources. The result of this step is a set of requirements that support the design and development of the solution. We also define two Design Objectives for the solution:

- Efficiency - The solution should process an optimisation task in a reasonable time span. It allows the end-user to swiftly receive an optimisation set that can be evaluated and implemented.
- Customisability - The solution should be customisable to meet the specific needs and requirements of different end-users.

The *design and development step* entails the actual development of the solution. The requirements of the previous step are taken into account during this step. The result is an artefact that can solve the problem and be evaluated. The *evaluation step* takes the developed artefact and tests its capabilities against a dataset. During this evaluation, results are produced that measure the solution's performance. In this thesis, the results represent the performance of the optimised models against the original model. The final step of the approach is the *communication step*. During this step, the results are discussed, and potential improvements for the next iteration of the solution are found. These improvements can then be converted into new requirements and fed back into the Design and Development step.

This thesis is structured as follows. Chapter 2 introduces the relevant background of business process models, simulation models and resource optimisation and discusses the related work relevant to optimisation and resources. Chapter 3 presents the problem definition, the multi-objective optimisation approach and discusses the development architecture and chosen technologies. In Chapter 4, we will describe the user interface developed for this thesis project. Chapter 5 empirically evaluates the optimisation approach with differentiated availability and interprets the findings of this thesis. Finally Chapter 6 concludes and sketches potential future work.

## 2. Background and related work

In this chapter, we describe and define the relevant concepts to set up the context for the research presented in this thesis. Besides this, we also describe the related work to Business Process Optimisation, Undifferentiated and Differentiated resources.

### 2.1. Business Process Model

A Business Process (**BP**) Model<sup>1</sup> is a graphical representation of a business workflow and its related sub-processes. This graphical representation exists out of:

- **Events** - An Event is something that “happens” during the course of a process. These events affect the flow of the model and usually have a cause (trigger) or an impact (result). E.g., start event, stop event.
- **Activities** - An Activity is a generic term for work that the company performs. E.g., ”Perform analysis”.
- **Gateways** - A Gateway is used to control the divergence and convergence of Sequence Flows in a Process. It will determine branching, forking, merging, and joining of paths E.g., ”Is the result positive?”
- **Sequence Flows** - A Sequence Flow is used to show the order that Activities will be performed in a Process.
- **Actors** - Actors or resources are the people involved in the process that are responsible for the activities performed. E.g., Clerk, Loan Officer

Most organisations have a good understanding of their business processes, however, not every process can be known to its deepest detail by every worker involved. A BP Model is a data-driven, visual representation that makes it possible for organisations to understand and optimise their workflows.

---

<sup>1</sup><https://www.omg.org/bpmn/>

## 2.2. Simulation Model with Differentiated Resources

The simulation model we will use for this thesis is a BP simulation model with differentiated resources. In this simulation model, the notion of *resource pool* is replaced with that of *resource profile*. Similar to a resource pool, a resource profile models a set of resources that share the same availability calendar. In these simulation models, an activity in a process model may be assigned to multiple resource profiles, and the same resource profile may be shared by multiple pools. For example, in a claims handling process, there may be a resource profile for *junior claims handler*, another for *senior claims handler* and a third for *lead claims handler*, each with different calendars. Activity *Analyse claim* may be assigned to *junior claims handler* and *senior claims handler*, i.e., an instance of *Analyse claim* may be performed by a junior or by a senior claims handler. Meanwhile, activity *Assess claim* may be assigned to *senior claims handler* and *lead claims handler*. Finally, activity *Approve large claim* may be assigned to *lead claims handler*, i.e., only lead claims handlers may perform this activity. Another difference is that in a classic simulation model, each activity is mapped to a distribution of processing times. Meanwhile, in a simulation model with differentiated resources, the distribution of processing times depends not only on the activity but also on the resource profile. Thus, the distribution of processing times of the activity *Analyse claim* when assigned to a *junior claims handler* is different than when assigned to a *senior claims handler*, e.g., seniors may be faster, on average, than juniors.

### Definition 1 (BP simulation model with differentiated resources).

A BP simulation model with differentiated resources DSM is a tuple  $\langle E, A, G, F, \text{RPROF}, BP, AT, AC \rangle$ , where  $E, A, G$  are the sets of events, activities, and gateways of a BPMN model,  $F$  is the set of directed flow arcs of a BPMN model, and the remaining elements capture simulation parameters as follows:

1.  $\text{RPROF} = \{r_1, \dots, r_n\}$  is a set of resource profiles, where  $n$  is the number of resources in the process, and each resource  $r \in R$  is described by:
  - $\text{ALLOC}(r) = \{\alpha \mid \alpha \in A\}$  is the set of activities that  $r$  can execute,
  - $\text{PERF}(r, \alpha) = R \times A^m \rightarrow \mathcal{P}^m(\mathbb{R}^+)$  is a mapping from the resource  $r$  to a list of density functions over positive real numbers,

corresponding to the distribution of processing times of each activity  $\alpha \in \text{ALLOC}$ , with  $m$  being the number of activities that  $r$  can perform,

- $\text{AVAIL}(r)$  is the calendar (a set of intervals) in which the resource  $r$  is available to perform each activity  $\alpha \in \text{ALLOC}$ ,
- $\text{COST}(r)$  is the cost of the resource  $r$  per time unit (e.g., hour)
- $PT : A \rightarrow \mathcal{P}(\mathbb{R}^+)$  is a mapping from each activity  $a \in A$  to a probability density function, modelling the processing times of activity  $a$ .
- $\text{BP} : F \rightarrow [0, 1]$  is a function that maps each flow  $f \in F$  s.t., the source of  $f$  is an element of  $G$  to a probability (a.k.a., the branching probability).
- $\text{AT} \in \mathcal{P}(\mathbb{R}^+)$  is a probability density function modelling the inter-arrival times between consecutive case creations.
- $\text{AC}$  is a calendar (set of intervals) such that cases can only be created during an interval in  $\text{AC}$ .

### 2.3. Pareto Fronts and Meta-Heuristic Optimisation Algorithms

In  $N$ -dimensional space, a solution  $S_2$  is Pareto dominated by another solution  $S_1$  if  $S_1$  is better than  $S_2$  for at least one objective, and  $S_1$  is at least as good as  $S_2$  for the remaining objectives [2]. For example, consider the two-dimensional space formed by the cycle time and cost from the execution of a business process  $P$ . Let us assume that the pairs (cycle time, cost) from the different resource allocations  $R_1$ ,  $R_2$  and  $R_3$  over  $P$  are on average  $P(R_1) = (20\text{hours}, 50\text{euros})$ ,  $P(R_2) = (15\text{hours}, 80\text{euros})$  and  $P(R_3) = (10\text{hours}, 45\text{euros})$ . Under minimisation constraints,  $P(R_1)$  and  $P(R_2)$  are Pareto dominated by  $P(R_3)$ , indicating that the resource allocation  $R_3$  is the best among all.

The set of solutions not dominated by any other is called Pareto optimal. The set of non-dominated points is called the Pareto set, and the evaluation of the objective functions on those points constitutes the Pareto front[2]. For example, assume that resources in  $R_3$  had a salary rise, which increases the execution costs so that  $P(R_3) = (10\text{hours}, 100\text{euros})$ . After the salary rise,  $R_3$  leads to the lowest cycle time with the highest cost among the three

allocations. On the contrary,  $R_1$  offers the lowest price but the worse cycle times. Meanwhile,  $R_2$  costs less than  $R_3$  and has a shorter cycle time than  $R_1$ . Thus, none of the three allocations dominates the other two. In this case, we do not have a single optimal allocation, but a Pareto set  $\{R_1, R_2, R_3\}$  of optimal allocations, which produces the Pareto front  $\{P(R_1), P(R_2), P(R_3)\}$  from executing the process  $P$ . Resource allocation is a widely recognised NP-complete problem, meaning that an efficient solution is impossible due to the impracticality of exploring the entire solution space. As a result, meta-heuristic algorithms can be employed to approximate Pareto fronts.

There are many ways to classify meta-heuristic optimization algorithms, and one of them is to group them into two categories: single-solution-based and population-based. Single-solution algorithms involve maintaining one solution and looking for better solutions through a perturbation function at each step. Population-based algorithms, on the other hand, maintain a population of solutions and generate a new population at each step by perturbing and merging solutions from the current population. Although single-solution methods are more efficient and explore fewer solutions, population-based techniques yield more optimal solutions by exploring a larger number of solution candidates. This observation has been reported in [3].

Hill-climbing is an optimisation technique that performs a search within the local vicinity of a given point. During each iteration, the algorithm identifies the optimal point to move to within the current neighbourhood of points. As a result, the algorithm enhances the current solution at each iteration, except when there are no superior solutions present within the entire neighbourhood. The conventional use cases of this algorithm, as documented in [3], typically involve a solitary objective, such as time, cost, or a linear combination of the two. The authors of [4] introduce a modified version of hill-climbing that can be used for multi-objective optimisation and to compute a Pareto front. Rather than storing a single solution, the algorithm saves a Pareto front. This enables the generation of new solution candidates by exploring the neighbourhood of each point in the current front. While the greedy nature of hill-climbing enables it to converge quickly, the algorithm may get stuck at a local optimum.

## 2.4. Resource optimisation in business processes

Several studies have addressed the problem of optimising the allocation of resources in a business process, i.e. the problem of determining how many

resources should be allocated to perform the activities in the process. Most studies in this field approach this problem as a single-objective optimisation problem, i.e., either by optimising one performance measure or combining several into a linear function [5, 6, 7, 8]. In [6], the authors proposed an evolutionary algorithm that takes as input a Petri Net and performs simulations to optimise the cycle time and resource cost, combined via a linear function (i.e. a weighted sum of cycle time and resource cost). Similar approaches using genetic algorithms and simulation models on single-objective problems were presented in [9, 10]. Another study addressing the allocation problem as a single-objective optimisation problem is presented in [5]. The approach minimises the resources constrained by a specified maximum waiting time. This thesis adopts a different approach. It addresses the problem of resource allocation as a multi-objective optimisation problem. Here, we aim to discover not a single but a set of optimal solutions, which allows process analysts to explore the available trade-offs between cycle time and cost.

In [11], the authors use a grid-search approach, i.e., an exhaustive exploration of all possible resource allocations given a minimum and a maximum number of resources per pool. This approach can be applied to explore the resource allocation space when the number of pools is small. However, it does not scale up to larger search spaces.

The problem of design-time resource allocation tackled in the presented research is related to the problem of runtime scheduling and runtime assignment of resources to work items in a business process. The latter issues have been tackled in various previous studies. For example, [4] and [12] consider the problem of deciding how to schedule the work items generated by each execution of a business process, taking into account that resources have availability constraints (i.e., they are available at some times but not at others). Meanwhile, [13] tackles the problem of deciding which specific resource should be assigned to a given work item, given the characteristics of each resource. The contribution of the present paper and those of the above papers are complementary.

The closest work related to the optimisation approach proposed in this thesis is a paper on resource allocation presented in [14]. This latter proposal utilises hill-climbing and tabu search meta-heuristics to retrieve the set of optimal resource allocations that simultaneously minimise cycle time and cost. However, the approach relies on simulations over pooled allocations of undifferentiated resources. Thus, due to the missing information on each independent resource, the optimisation approach is limited to only increasing

and decreasing the size of the pools, i.e., to mitigate high and low resource utilisation.

In contrast, this thesis proposes a new set of meta-heuristics over a hill-climbing search that exploits the characteristics of each independent resource over a differentiated model. Unlike [14], the presented research not only adds or removes resources but also updates the availability calendar of each resource to find resource configurations that jointly optimise cost and cycle time in the setting where resources have differentiated availability calendars.

### 3. Design and Architecture

This chapter is split into two sections. First, we will discuss the problem definition, explaining Pareto fronts and the objective functions. The second part is focused on the architecture and deployment of the web application from a developer’s viewpoint and the relevant technologies.

#### 3.1. Resource Allocation and Roster Optimisation with Differentiated Resources

This section considers the problem of jointly optimising the cost of resources and the cycle time of a process by perturbing the allocation of resources to activities in a process (a.k.a. the resource allocation) and the availability calendars of the resources (a.k.a. the roster), in the setting where each resource has its own calendar, but all resources share the same performance with respect to any given activity.

In this context, an ideal resource configuration is one that minimises both the resource cost and the cycle time. Typically, no single configuration exists that minimises time and cost simultaneously. Instead, there is a set of (incomparable) optimal configurations (a.k.a. Pareto front) so that for each configuration in this Pareto front, no performance measure, e.g., time and cost, can be improved without scarifying any other [14]. Accordingly, this chapter considers the problem of computing a Pareto front of resource configurations with respect to resource cost and cycle time, where a resource configuration consists of a resource allocation (i.e., a set of resources together with a relation between resources and the activities they can perform) and a resource roster (i.e., a mapping from resources to availability calendars).

We note that the fewer resources available, i.e., the amount of time dedicated to working, the lower the waiting times, and thus the cycle times, will be. Similarly, the more resources are available, the less busy they are, i.e., lower resource utilisation, leading to decreasing waiting times and, thus, decreasing cycle times. However, increasing the number of resources or their calendar unavoidably increases the cost of the hours worked by the resources collectively. Conversely, reducing the number of resources or shrinking their availability calendars is likely to decrease the resource costs at the expense of increasing waiting times due to higher resource utilization [15].



### 3.1.1. Problem Definition: Pareto Front and Objective Functions

**Definition 2 (Optimisation Problem).** *Given a simulation model with differentiated resources  $SM = \langle E, A, G, F, RPROF, BP, AT, AC \rangle$  according to Def. 1, the optimisation problem consists of finding the Pareto front of resource profiles  $RPROF$  that minimises the overall process cycle time and cost, that is:*

- **minimize**  $\text{cycle\_t} = \frac{\sum_{t \in L} [\max\{\tau_c \in t\} - \min\{\tau_s \in t\}]}{|L|}$ , i.e., minimize the average cycle time of all traces  $t$  in an event log  $L$  representing the execution of the model  $SM$ . The cycle time of each trace  $t$  is the difference between the timestamps  $\tau_c, \tau_s$  corresponding to the last event completed and the first event started in  $t$ , respectively.
- **minimize**  $\text{cost}_p = \sum_{r \in RProf} \text{Cost}(r) \times |\text{AVAIL}(r) \cap [\min\{\tau_s \in L\}, \max\{\tau_c \in L\}]|$ , i.e., minimise the total cost of all the available resources during the process execution. The total cost of each resource  $r$  is the whole time they were available in the interval spanned by the timestamps  $\tau_s, \tau_c$  of the first/last events started/completed in an event log  $L$  representing the execution of the model  $SM$  multiplied by their cost per time unit (e.g., hour).

The key difference in Definition 2, compared to undifferentiated simulation models is that the notion of  $R\text{Pools}$ , i.e., Resource Pools, is replaced by  $R\text{Profs}$ , i.e., Resource Profiles. As described in Definition 1, differentiated resources have their own resource profile. This change does not affect the formula calculating the cycle time or cost but does affect the end result of the formula. Since we do not just count the number of resources in a pool and calculate the cost of that pool, but calculate the cost of the individual resources, which in the case of differentiated resource models are not part of a "pool".

Adjusting the availability calendars as part of the roster optimisation problem is typically restricted by several constraints dictating which intervals to extend or shorten over the resource calendars. For example, in the presence of human resources, they usually work a maximum of 8 hours daily or a maximum of 40 hours a week. Besides, there are time slots in which some resources must always work or never work to satisfy the logistics of the whole organisation. Accordingly, Definition 3 formalises a set the optimisation constraints allowed by the optimisation approach presented in this paper.

**Definition 3 (Optimisation Constraints).** Consider the weekly calendar  $\widehat{C}_r$  of each resource  $r$ , i.e., specified by the corresponding AVAIL function in RPROF. Let  $\widehat{C}_r^d \subseteq \widehat{C}_r$ , with  $d \in W = \{\text{Monday}, \dots, \text{Sunday}\}$ , be the availability calendar of the resource  $r$  in week-day  $d$ , e.g.,  $\widehat{C}_r^{\text{Monday}}$  corresponds to the working times of  $r$  every Monday. The optimisation problem defined in 2 must satisfy the following constraints:

- $\forall \kappa = \langle \omega, \delta \rangle \in \widehat{C}_r, \exists \text{Min}_\delta, \alpha \in \mathbf{N}^+ : |\delta| \geq \text{Min}_\delta \wedge |\delta| = \alpha \times \text{Min}_\delta$ , i.e.,  $\text{Min}_\delta$  is the (minimum) granule size so that the duration of the intervals  $\delta = \langle \tau_s^w, \tau_c^w \rangle$  in each calendar entry  $\kappa \in \widehat{C}_r$  is a multiple of  $\text{Min}_\delta$ ,
- $\forall r \in RProf, \exists \eta_r \in \mathbf{N}^+ : |\widehat{C}_r| \leq \eta_r$ , i.e.,  $\eta_r$  is the maximum capacity (measured in time units) that each resource  $r$  can work on a week, a.k.a., weekly-capacity. Analogously,  $\eta_r^d \in \mathbf{N}^+, |\widehat{C}_r^d| \leq \eta_r^d$ , specifies the maximum capacity of the resource  $r$  in a single day, a.k.a., daily-capacity,
- $\forall \delta \in \widehat{C}_r^d, |\delta| \leq \text{Max}_\delta^d, \text{Max}_\delta^d \in \mathbf{N}$ , i.e.,  $\text{Max}_\delta^d$  is the maximum number of consecutive time units a resource can work in a day,
- $\widehat{C}_r^-$ , with  $\widehat{C}_r^- \cap \widehat{C}_r = \emptyset$ , a.k.a., the never-working calendar, specifies the intervals of each day in which the resource  $r$  never works,
- $\widehat{C}_r^+$ , with  $\widehat{C}_r^+ \cap \widehat{C}_r = \widehat{C}_r^+$ , a.k.a., the always-working calendar, specifies the intervals of each day in which the resource  $r$  always works.

The problem of roster optimisation addressed in this paper is a variant of the resource allocation problem, which is a well-known NP-complete problem. In other words, no efficient solution exists, i.e., exploring the entire solution space is impossible in practical scenarios. Instead, meta-heuristic algorithms to reduce the search space are used to approximate Pareto fronts. Specifically, in this paper, we focus on the tasks and resources that exhibit poor results in terms of execution costs as well as waiting and idle times to heuristically decide which calendars to improve as part of a hill-climbing search.

### 3.1.2. Discovering Optimal Rosters: Hill-Climbing Meta-heuristic

The optimisation approach presented in this paper follows a well-known meta-heuristic named Hill-climbing search. Hill-climbing is an iterative optimisation technique that performs a local search around a given solution

candidate. At each iteration, the algorithm explores the neighbourhood of the current solution candidate, intending to find a better solution to be explored in the next iteration. Therefore, the algorithm improves the current solution at each iteration and stops after exploring a neighbourhood that does not yield any better solution than the ones already in the Pareto front. The classic hill-climbing algorithm focuses on single objective optimisation [3] (e.g., time, cost, or a linear combination of both). However, in [16], the authors extended the algorithm for multi-objective optimisation, meaning that the algorithm returns a Pareto front of solutions instead of a single solution. In this approach, new solution candidates are generated by exploring the neighbourhood of each solution in the current Pareto front.

In the scope of roster allocation, exploring the neighbourhood of each solution in the Pareto front is too broad. For example, updating the availability calendar  $\widehat{C}_r^+$  of a single resource produces (at least)  $2^k$  solutions candidates, where  $k$  is the number of calendar entries. The latest number considers only two operations, i.e., expanding and shrinking the time intervals, although many others can be performed, increasing this number significantly. Note that those potential solution candidates relate to a single resource. Spread that to all the resources in each roster and all the rosters in the Pareto front. Then, searching the neighbourhood space becomes unfeasible in practical scenarios, especially when the number of resources is high.

In the following, we propose an approach to prune the hill-climbing searching space to discover the Pareto front with the optimal rosters. Instead of exploring the entire neighbourhood at each iteration, the proposal heuristically selects which resources have a potentially higher (negative) impact on cost or cycle time. Specifically, Algorithm 1 incorporates five perturbations that address different possible causes of cycle times or cost inefficiency by perturbing the calendars of the resources that are involved in these inefficiencies.

---

**Algorithm 1** Perturbation Method: Generating the Pareto front
 

---

```

1: function EXPLORESOLUTIONSPACE(SM: Simulation Model,  $P$ : RPROF)
2:    $PSet \leftarrow \{RP\}$ 
3:    $Q \leftarrow \text{PRIORITYQUEUE}(\{P\})$ 
4:   while  $Q \neq \emptyset$  do
5:      $P \leftarrow \text{POP}(Q)$ 
6:      $\triangleright$  Perturbation 1: Updating availability calendars to reduce waiting times
7:      $wt\_list \leftarrow \text{SORTGRANULESTOIMPROVEWAITINGTIMES}$ 
8:     for each  $\langle r, \kappa \rangle \in wt\_list$  do
9:       if  $\text{SCANPARETO}(Q, PSet, SM, [L^+(P, \hat{C}_r, \kappa), \ll(P, \hat{C}_r, \kappa)])$  then
10:        break
11:      $\triangleright$  Perturbation 2: Updating availability calendars to reduce costs
12:      $c\_list \leftarrow \text{SORTGRANULESTOIMPROVECOST}$ 
13:     for each  $\langle r, \kappa \rangle \in c\_list$  do
14:       if  $\text{SCANPARETO}(Q, PSet, SM, [LR^-(P, \hat{C}_r, \kappa), L^-(P, \hat{C}_r, \kappa), R^-(P, \hat{C}_r, \kappa)])$  then
15:        break
16:      $\triangleright$  Perturbation 3: Updating availability calendars to reduce idle times
17:      $it\_list \leftarrow \text{SORTGRANULESTOIMPROVEIDLETIMES}$ 
18:     for each  $\langle r, \kappa \rangle \in it\_list$  do
19:       if  $\text{SCANPARETO}(Q, PSet, SM, [R^+(P, \hat{C}_r, \kappa), >>(P, \hat{C}_r, \kappa)])$  then
20:        break
21:      $\triangleright$  Perturbation 4: Adding resources to reduce waiting times
22:      $tk\_list \leftarrow \text{SORTTASKSTOIMPROVEWAITINGTIMES}$ 
23:     for each  $t \in tk\_list$  do
24:       if  $\text{SCANPARETO}(Q, PSet, SM, [P \cup \{min\_cost(P, t)\}])$  then
25:        break
26:      $\triangleright$  Perturbation 5: Removing resources to reduce costs
27:      $r\_list \leftarrow \text{SORTRESOURCETOIMPROVECOST}$ 
28:     for each  $r \in r\_list$  do
29:       if  $\text{SCANPARETO}(Q, PSet, SM, [P \setminus \{r\}])$  then
30:        break
31:   return  $PSet$ 

```

---

**Perturbation 1** explores how to decrease/remove waiting time inefficiencies during the process execution. To that end, the function `SORTGRANULESTOIMPROVEWAITINGTIMES` in line 6 filters the tasks affected by waiting times. Then it sort and retrieves in descending order the pairs resource-calendar entries  $(r, \kappa)$  with a higher incidence at the intervals where the waiting times occurred during the process execution/simulation from each affected task. Lines 7-9, iteratively from worst-best, try to fix the calendar entries marked as problematic, stopping at the entry with the highest impact that can be improved. Here, the issue consists of a task enabled to be executed while the resource is unavailable. Then, the correction aims to make the resource available earlier in the problematic calendar entry by decreasing the lower bound of the corresponding time granule using two operations (Figure 2 b) and h)). The first operation  $L^+$  takes the granule  $\kappa = \langle \omega, \tau_s^w, \tau_c^w \rangle \in \hat{C}_r$  and expands the

interval by subtracting the minimum granule size from its lower bound, i.e.,  $\kappa' = \langle w, \tau_s^w - Min_\delta, \tau_c^w \rangle$ . The second operation  $\ll$  takes the granule  $\kappa = \langle \omega, \tau_s^w, \tau_c^w \rangle \in \widehat{C}_r$  and shifts the entire interval to the left, keeping its original size, by subtracting the minimum granule size from its lower and upper bounds, i.e.,  $\kappa' = \langle w, \tau_s^w - Min_\delta, \tau_c^w - Min_\delta \rangle$ .

**Perturbation 2** focuses on reducing costs during the process execution. The function `SortGranulesToImproveCost` in line 10 filters and sorts, in descending order, the pairs resource-calendar entries  $(r, \kappa)$  of the resources according to the total costs incurred during the process execution. Lines 11-13 iteratively attempt the sorted calendar entries stopping at the one with the highest cost that can be improved. In this perturbation, the issue may relate to resources working more hours than required to fulfill their duties during the process execution. Intuitively, the adjustment reduces resource availability by decreasing the calendar entry using three operations (Figure 2 g), c), and e)). The first operation  $LR^-$  takes the granule  $\kappa = \langle \omega, \tau_s^w, \tau_c^w \rangle \in \widehat{C}_r$  and shrinks the interval by adding the minimum granule size to the lower bound and subtracting from the upper, i.e.,  $\kappa' = \langle w, \tau_s^w + Min_\delta, \tau_c^w - Min_\delta \rangle$ . The operations  $L^-$  and  $R^-$  take smaller steps by only reducing, respectively, the lower and upper bounds, i.e., producing  $\kappa' = \langle w, \tau_s^w + Min_\delta, \tau_c^w \rangle$  and  $\kappa' = \langle w, \tau_s^w, \tau_c^w - Min_\delta \rangle$ , respectively.

**Perturbation 3** aims to decrease/remove idle time inefficiencies during the process execution. The function `SortGranulesToImproveIdleTimes`, in line 14, descending sorts and retrieve the pairs resource-calendar entries  $(r, \kappa)$  similarly to `SortGranulesToImproveWaitingTimes` in perturbation 1, but considering idle times instead. Then, lines 15-17 iteratively try fixing the entries until finding one improving the Pareto front. Here, the root cause consists of resources becoming unavailable during the execution of the started tasks, which delays their completion. Then, the correction aims to increase resource availability in the problematic calendar entry by raising the upper bound of the corresponding time interval using two operations (Figure 2 d) and i)). The first operation  $R^+$  takes the granule  $\kappa = \langle \omega, \tau_s^w, \tau_c^w \rangle \in \widehat{C}_r$  and expands the interval by adding the minimum granule size to its upper bound, i.e.,  $\kappa' = \langle w, \tau_s^w, \tau_c^w + Min_\delta \rangle$ . The second operation  $\gg$  takes the granule  $\kappa = \langle \omega, \tau_s^w, \tau_c^w \rangle \in \widehat{C}_r$  and shifts the entire interval to the

right, keeping its original size, by adding the minimum granule size to its lower and upper bounds, i.e.,  $\kappa' = \langle w, \tau_s^w + Min_\delta, \tau_c^w + Min_\delta \rangle$ .

**Perturbation 4** again focuses on waiting time inefficiencies arising from high resource utilization, keeping resources busy to perform the tasks when they become available. The latest may be an indicator that new resources are required. To decide where to add them, function `SortTasksToImproveWaitingTimes`, in line 18, sorts the tasks according to their waiting times. Then, lines 19-21 attempt to improve the most problematic task by adding a new resource profile that can execute it. Like in perturbations 1-3, the correction tries all the tasks facing waiting times issues iteratively until finding one that can be fixed. Adding a resource may increase the overall process costs. Thus, to reduce the impact of that issue, the algorithm copies the resource profile with lower costs among those who can execute the corresponding task, i.e., described by  $mincost(P, t)$  in line 20.

**Perturbation 5** focuses on reducing the overall process cost. In contrast to perturbation 4, this perturbation aims at reducing cost by increasing the utilization of potentially under-utilized resources (i.e. more resources are available than required). The underlying idea is that removing underutilized resources may decrease the overall costs without strongly affecting cycle time. Accordingly, the function `SortResourcesToImproveCost` in line 22 filters and descendant sorts the resources according to their cost and utilization. Then, lines 23-25 follow the same iterative approach to remove the resource with the highest impact possible, i.e., represented by  $P \setminus \{r\}$  in line 24.

Figure 2 graphically illustrates the possible transformations over a single time granule  $\delta$  in a calendar entry  $\kappa$  used by perturbations 1-3 in Algorithm 1. All the operations receive an entry  $\kappa$  from the calendar of the  $\hat{C}_r$  of one resource  $r$  in the set of resource profiles  $P$ . Then, they modify the calendar of  $r$ , “updating and” returning the resource profiles  $P$  accordingly. Indeed, those operations enforce the “optimisation constraints” presented in Definition 3. Thus, the functions never produce calendars exceeding the maximum daily/weekly resource capacity, never remove entries from the “always working” calendar  $\hat{C}_r^+$ , and they do not add entries to the “never working” calendar  $\hat{C}_r^-$  of the resource.

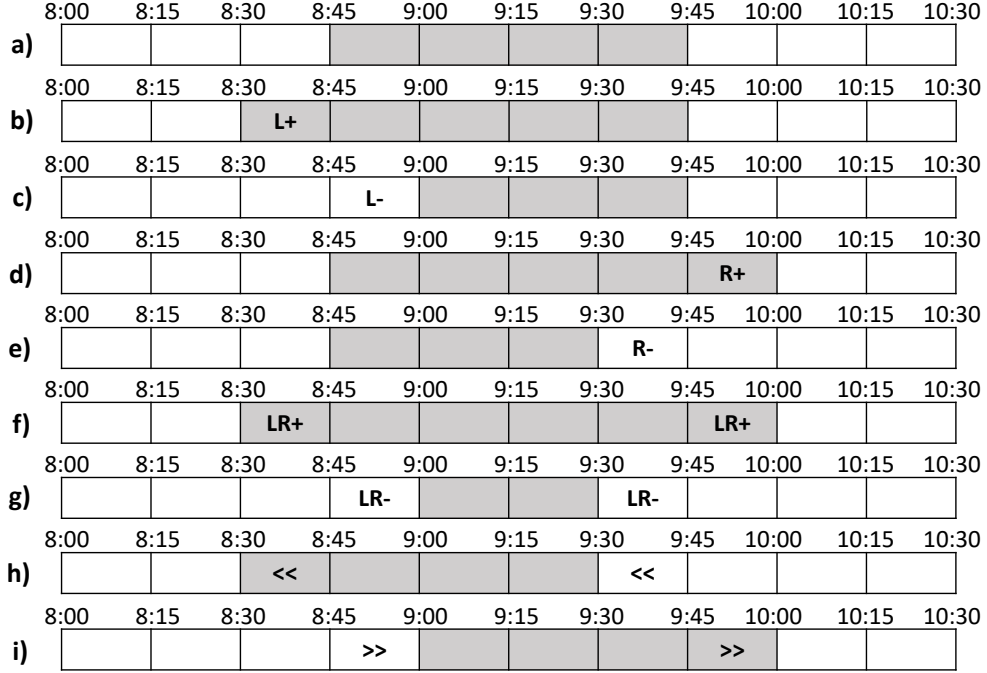


Figure 2: Operations to modify the time interval defined by the granule  $\kappa$  (on top). a) shows the original time interval, b) to i) the resulting intervals after applying the corresponding operation over a).

Algorithm 1 stores in a priority queue  $Q$  any solution candidate which improved the Pareto front after applying any of the transformations described by perturbations 1-5. At each iteration, the resource profile  $P$  on top of the queue is perturbed, aiming to improve the current Pareto front. Note that  $Q$  may contain optimal resource profiles in a given iteration but overtaken by others in subsequent iterations. In those cases, the queue sorts and retrieves the overtaken profiles according to their euclidean distance to the current Pareto front. The algorithm stops when the queue is empty, retrieving the optimal resource profiles stored in  $PSet$ . Other stopping criteria are possible, e.g., the maximum number of iterations or candidates generated. However, for simplicity, we omitted them.

Algorithm 1 relies on the function `SCANPARETO` to assess if a transformation described by perturbations 1-5 improves the Pareto front. Algorithm 2 outlines the assessment using simulations. It receives as input the priority queue  $Q$ , the current Pareto set  $PSet$ , the simulation model  $SM$ , and the resource profiles candidates obtained after each perturbation. Then, the pro-

---

**Algorithm 2** Updating Pareto Set using process simulations

---

```
1: function SCANPARETO( $Q$ ,  $PSet$ ,  $SM$ ,  $RProfile\_candidates$ )
2:    $isImproved \leftarrow \text{False}$ 
3:   for each  $P \in RProfile\_candidates$  do
4:      $sLog \leftarrow \text{SIMULATEPROCESS}(SM, P)$ 
5:     if  $isNONDOMINATED(P, PSet, sLog)$  then
6:        $FIXPARETO(PSet, P)$ 
7:        $ENQUEUE(Q, P)$ 
8:        $isImproved \leftarrow \text{True}$ 
9:   return  $isImproved$ 
```

---

cess model is simulated (line 4) for each candidate to estimate the resulting cycle times and cost. Suppose the new candidate is not dominated by any of the profiles in the current Pareto set (line 5). In that case, the new Pareto set is updated, e.g., possibly removing previous solutions after adding the new profile (line 6). Similarly, the distances of the removed solutions are updated in the priority queue  $Q$  (line 7). Algorithm 2 returns whether any solution candidate improved the optimal rosters.

## 3.2. Architecture and Deployment

In this section, we will give an in-depth overview of the application architecture and its components as well as the deployment environment. First, we will describe the high-level architecture of the application, after which we will delve deeper into each tier and its components. Besides this, we will also describe the technologies used to develop the application and the reasoning behind their selection. Afterwards, we will discuss the deployment environment of the application and the related technology. Finally, we will close this section by providing a summary.

### 3.2.1. Architecture Overview

Optimos is designed using a well-established software architecture, namely the Three-Tier Architecture. This architecture separates the application into three layers, the Presentation Tier, the Application Tier and the Data Tier. The main benefit of this architecture is that each tier consists of its own infrastructure and can be developed, updated and scaled simultaneously without impacting the other tiers. Figure 3 illustrates the high-level overview of this architecture with the chosen technologies for each Tier.



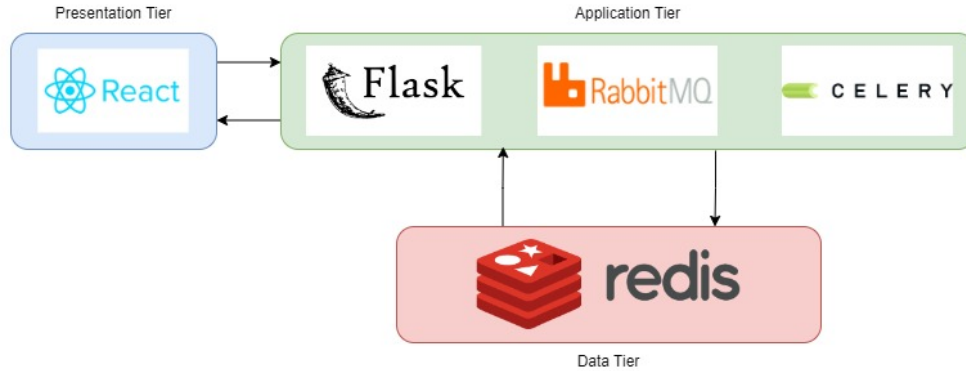


Figure 3: High-level architecture of Optimos.

*Presentation Tier.* The presentation tier contains the user interface and is the main point of interaction between the end user and the application. The structure of this tier can vary depending on the application. However, the purpose of this tier stays the same, namely:

- Display information from the end user
- Collect information from the end user

This thesis implements the presentation tier through a web interface using React Typescript. A JavaScript library for building user interfaces.

**React**<sup>2</sup> is an open-source library that allows developers to build complex applications with reusable UI components. This thesis uses an adaptation of React, namely React Typescript. TypeScript is a statically-types superset of JavaScript that adds features like strong typing, where each program variable must have a datatype predefined by the programming language. React TypeScript allows developers to improve code maintainability and increase productivity.

**React** was chosen as the technology in this thesis because of its larger community, out-of-the-box components and much richer ecosystem than other comparable libraries, like Vue or Angular. Another reason is that Optimos is part of a more extensive toolset, currently being developed for the **Process**

---

<sup>2</sup><https://react.dev/>

**Improvement eXplorer (PIX)**<sup>3</sup> project. It was decided that the toolset would use React as the standard library for front-end development.

*Application Tier.* The application tier contains the core of the application. This is also the most complex tier of the three tiers, as the full functionality of the application depends on it. The main purpose of this tier is:

- Process incoming data
- Handle incoming commands and requests
- Perform logical decisions and execute processes
- Communicate between the Data tier and Presentation tier

The application tier of this thesis utilises three distinct technologies, specifically PyFlask, RabbitMQ, and Celery, to implement various functionalities. We will implement the application tier using the Master-Slave architecture. This architecture is a common approach to parallelise workloads, making it possible to handle large-scale computing tasks more efficiently. To take this one step further we will extend the architecture to work asynchronously using a message broker like RabbitMQ. Figure 4 illustrates the Master-Slave architecture implementation in our application.

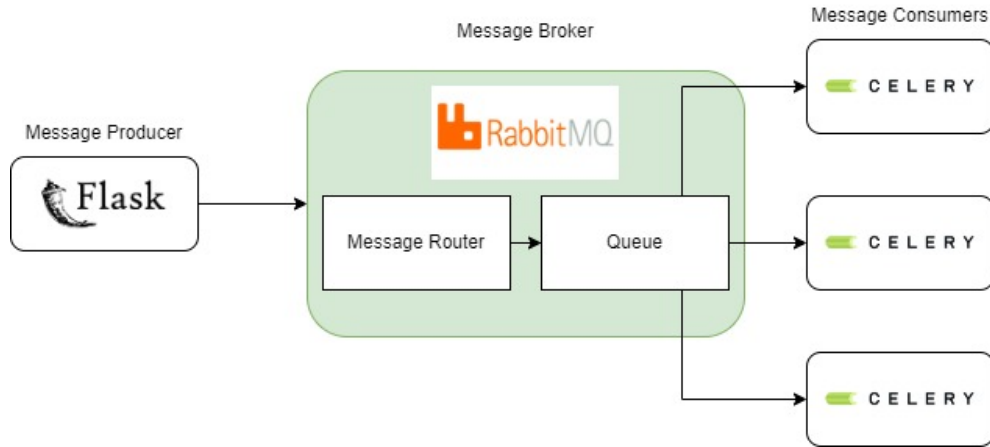


Figure 4: Master-Slave Architecture used in the Application Tier

---

<sup>3</sup><https://cordis.europa.eu/project/id/834141>

**PyFlask** or **Python Flask**<sup>4</sup> is a Python-based, open-source, lightweight framework that provides a wide range of features for web development, including routing, templating and session management. Within the scope of the thesis, PyFlask is used to develop the API endpoints of the application. PyFlask was chosen as technology because we wanted to reduce the number of programming languages used during development. The back-end system is developed in Python, so a natural continuation from this would be to implement the API in PyFlask, reducing the complexity of data transmission yet building a reliable and robust service.

**RabbitMQ**<sup>5</sup> is an open-source message broker that enables a fast exchange between applications and systems. In this thesis, we use RabbitMQ as a message broker because it allows applications to communicate asynchronously using the Advanced Message Queueing Protocol (AMQP). Another reason is that RabbitMQ can distribute tasks between the multiple Celery workers, allowing the application to scale easily.

**Celery**<sup>6</sup> is a Python-based, open-source, task queueing software whose primary goal is to execute processes asynchronously. The optimisation approach of the application is computationally intensive and requires considerable time to execute over larger processes. This would hamper the application's functionality as the system would be blocked until the optimisation has finished. Celery allows us to execute these tasks asynchronously without compromising the system. Even though Celery is not the only task queue software, it is reliable and is also Python-based. Thus, reducing the complexity again.

*Data Tier.* The data tier is responsible for storing and managing all the relevant data to the application. This is usually in the form of a database or a file repository. In this thesis, we implemented the data tier as a database using Redis.

**Redis**<sup>7</sup> is an open-source, in-memory data structure used as a database, cache and message broker. Redis' role in the application is as a central database. It stores nearly all the data provided by the presentation tier and

---

<sup>4</sup><https://flask.palletsprojects.com/en/2.2.x/>

<sup>5</sup><https://www.rabbitmq.com/>

<sup>6</sup><https://docs.celeryq.dev/en/stable/index.html>

<sup>7</sup><https://redis.io/>

generated by the application tier. We chose Redis as our database technology because it is a fast, reliable, scalable NoSQL database.

### 3.2.2. Deployment

The traditional application deployment process is rather arduous, leading to unexpected issues and time lost debugging the deployment environment. To combat this issue, a solution, commonly referred to as *Containerisation* was invented. Containerisation is, in essence, the packaging of application code with a set of operating systems libraries and dependencies that are required to run the code. This creates a group of isolated application packages that are easy to deploy, run and maintain. It also removes the need for maintaining the operating system environment used during the development by the developers. To facilitate the containerisation and deployment process of our application, we chose **Docker**.

**Docker**<sup>8</sup> is a software platform that makes it easy to deploy and run applications by using containers. Docker provides a wide range of tools that simplify the deployment of multi-container applications available for the developer.

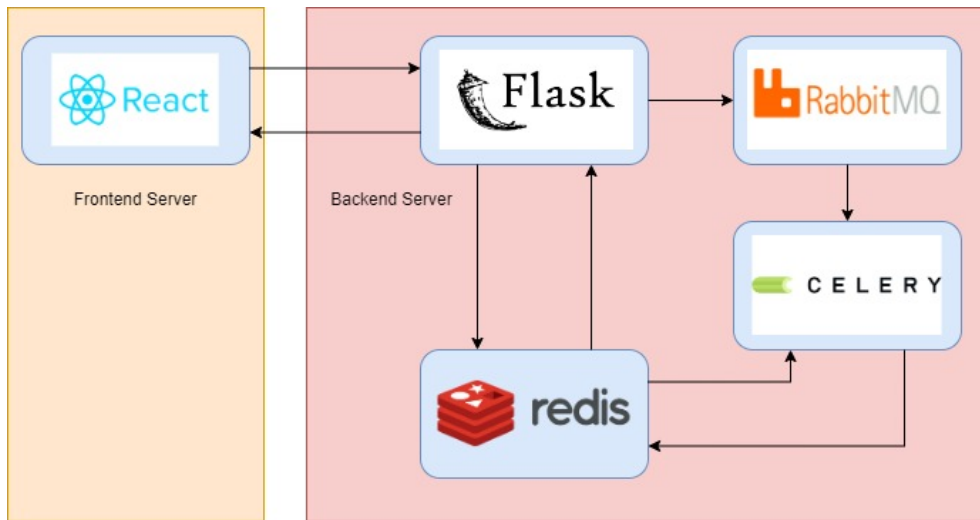


Figure 5: Live environment structure of the application.

---

<sup>8</sup><https://www.docker.com/>

*Live environment.* For the live deployment of the application, we opted for a two-server structure, with one server containing the presentation tier and the other server containing the application and data tier of the application. This way, the application tier is accessible even when the presentation tier is under maintenance. Figure 5 illustrates the structure of the live environment. The application is accessible on the following URL: <http://optimos.cloud.ut.ee> and is hosted on the servers of the University of Tartu.

In the appendix, we have added the Dockerfile 6 and docker-compose file 6 that were used to deploy the application on the live environments. All of the containers can communicate with each other using a bridge network that is automatically configured by Docker.

### 3.2.3. Architecture Summary

To close this chapter, table 1 summarises the technologies used and their role within the application.

| Technology | Role                | Purpose   |
|------------|---------------------|---|
| React      | User Interface      | Main interaction point  |
| PyFlask    | API service         | Communication facilitator between application and user interface      |
| RabbitMQ   | Message Broker      | Asynchronous communication facilitator between API and Celery workers |
| Celery     | Asynchronous worker | Optimisation task performer   |
| Redis      | Database            | Store all necessary data of the application                           |
| Docker     | Deployment          | Containerisation and live-environment deployment                      |

Table 1: Summary of used technologies

## 4. User Interface

This chapter presents the User-Interface (UI) of the developed application. First, we will discuss the application's structure from the developer's perspective, and then we will walk through the UI and describe each screen and its functionality.

### 4.1. UI Development

The previous chapter stated that the front end is developed using React TS. This library, paired with certain packages, allows developers to use off-the-shelf components that are ready for use but can be tailored to the user's specific needs. Combining this with a proper development workflow, we can mitigate many potential issues, which could cause reworks and take valuable time. To achieve this we decided to implement the front end using a Component-Based structure.

**Component** structure is a software architecture designed around building software systems from independent, interchangeable and reusable software components. Each component in this structure is designed to perform a specific function or set of functions and can communicate with other components through interfaces. This allows components to easily be added, removed or replaced without affecting the functionalities of other components. Secondly, as stated previously, Optimos is part of the PIX project, which already contains several generic, previously built components.

### 4.2. UI Walkthrough

In this section, we will walk through the user interface and its components from an end-user perspective. We will use an example, named "purchasing\_example" from the experiments performed in this thesis. This model describes the lifecycle of a purchase requisition process. We also want to clarify some terms used in this chapter:

- **Slot** - One unit of the defined granule in the constraint parameters. E.g., with a 60-minute granule, one slot is 60 minutes.
- **Shift** - One or more Slots combined together. Shifts have a clear start and end point and are split by at least one Slot. E.g., The clerk has a shift from 8.00 am until 11.00 am and from 2.00 pm until 7.00 pm.

- **Simulation Parameters** - A JSON file, containing the parameters required for the simulator to be able to execute the process simulations. More information about the format of the simulation parameters can be found on the GitHub repository of the simulator<sup>9</sup>
- **Constraint Parameters** - A JSON file, containing the parameters required for the optimiser to be able to execute the process optimisation. More information about the format of the constraint parameters can be found on the GitHub repository of the optimiser<sup>10</sup>

Both the simulation parameters and constraint parameters are an implementation of Definition 1 and Definition 3 respectively.

#### 4.2.1. Landing page

When users navigate to our application, they are greeted with the landing page, represented in Figure 6. On this page, users can upload several files necessary for an optimisation task. They can also use the Drag&Drop functionality to upload files more easily. A radio button, "Generate constraints", allows users only to upload the BPMN model and simulation parameters. The application will then generate constraints based on the simulation parameters given by the end user, which they can then adjust to their own requirements. When a user is ready to continue, and they have provided all of the required information, the button below the container will turn blue and can be clicked.

Note that the supported file extensions are *.zip*, *.json*, and *.bpmn*. We allow .zip files to be uploaded given they contain the right files for an optimisation task.

#### 4.2.2. Global Constraints page

After the files are validated and the entered information is correct, the global constraints tab is the first page visible to the user. This page is illustrated in Figure 7 and 8 Here they can enter details about the scenario, like:

- A unique scenario name.

---

<sup>9</sup><https://github.com/AutomatedProcessImprovement/Prosimos>

<sup>10</sup><https://github.com/AutomatedProcessImprovement/roptimus-prime>

Run optimization

Supported extensions: bpmn | json | zip

☐ Generate constraints?

BPMN Model

Simulation Parameters

Constraints Parameters

Drag and drop a file here or click

Figure 6: Landing page of Optimos

- The maximum number of iterations that can be performed before forcefully quitting the task.
- The preferred algorithm to use for the optimisation.
- The preferred approach the optimisation should follow.

Global Constraints Scen Consti

Scenario specification

Scenario name  
My first scenario

Total number of iterations  
100

Figure 7: Global Constraints tab - Name and iterations

The maximum number of iterations is an important limiter that directly impacts the time an optimisation task can take. If the maximum number of



iterations is a large number, e.g., 5000, it is likely the optimisation can take multiple hours to complete. This may ultimately lead to a more accurate set of optimal solutions but should be considered cautiously.

The different algorithms and approaches are described in Chapter 3 and Chapter 5.

The screenshot shows a web interface for the 'Global Constraints' tab. At the top, there are three tabs: 'Global Constraints' (active), 'Resource Constraints', and 'Simulation Results'. A 'DOWNLOAD SCENARIO FILES' button is located in the top right. Below the tabs, there are two dropdown menus. The first is labeled 'Algorithm' and has 'HC-FLEX | Hill Climb flex' selected. The second is labeled 'Approach' and has 'CO | CA/AR combined' selected.

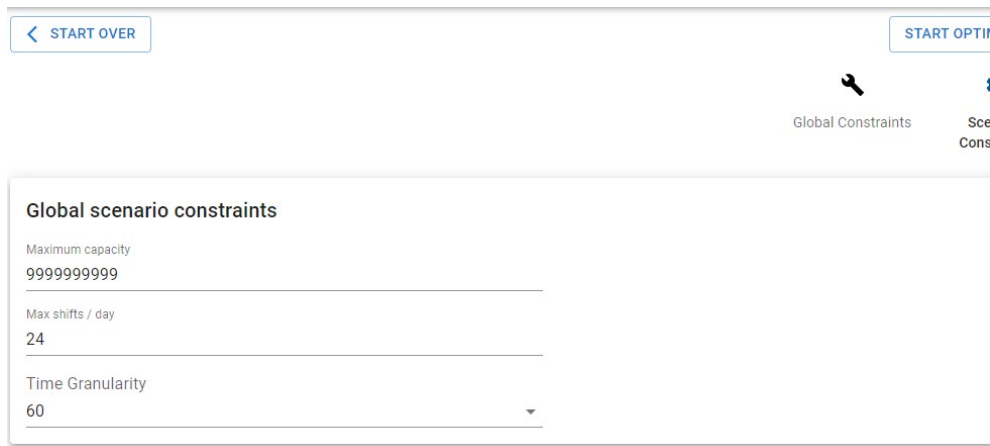
Figure 8: Global Constraints tab - Algorithm and approach

#### 4.2.3. Scenario Constraints page

The tab after the global constraints tab is the scenario constraints tab. In this tab, the user can define the following constraints that directly impact the overall limitations of the optimisation. This page is illustrated by Figure 9 and 10

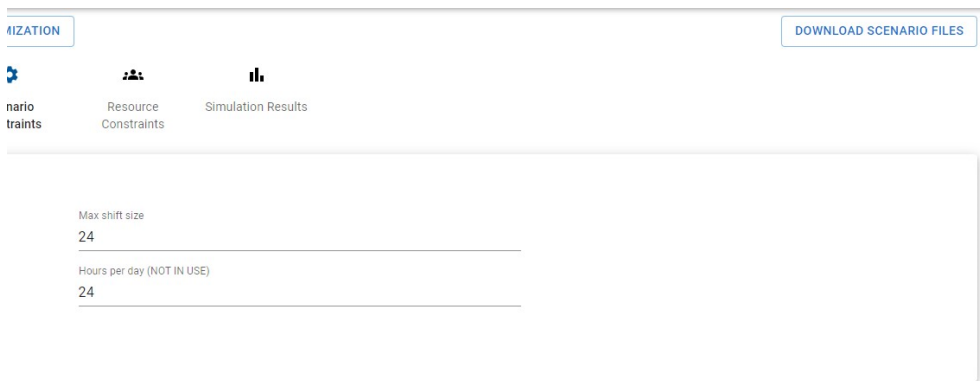
- Maximum capacity - This parameter represents the maximum sum of the resource times that can be performed per week by all resources that participate in the process combined. It limits the optimisation algorithm from exceeding a number set by the user.
- Max shifts/day - This parameter used to represent the maximum number of shifts resources could have assigned per day. This parameter has been moved to a lower level and can safely be ignored.
- Time granularity - This number represents the granularity of shift slots. E.g., 60 represents 1-hour shift slots. Meanwhile, 30 represents 30-minute shift slots. The granularity impacts the impact of the optimisation step from a time perspective.

- Max shifts size - This parameter is similar to max shifts/day and is being shifted to a lower level and can safely be ignored.
- Hours per day - This parameter limits the number of hours a day consists of. This parameter is not in use yet and safely be ignored.



The screenshot shows the 'Global scenario constraints' section of a web interface. At the top, there are two buttons: '< START OVER' on the left and 'START OPTIMIZATION' on the right. Below these, there are two links: 'Global Constraints' with a key icon and 'See Constraints' with a list icon. The main content area is titled 'Global scenario constraints' and contains three input fields: 'Maximum capacity' with the value '999999999', 'Max shifts / day' with the value '24', and 'Time Granularity' with a dropdown menu showing '60'.

Figure 9: Scenario Constraints tab - Capacity, shifts/day and granularity



The screenshot shows the 'Scenario Constraints' section of a web interface. At the top, there are three tabs: 'Scenario Constraints' (active), 'Resource Constraints', and 'Simulation Results'. To the right of the tabs is a button labeled 'DOWNLOAD SCENARIO FILES'. Below the tabs, there are two input fields: 'Max shift size' with the value '24' and 'Hours per day (NOT IN USE)' with the value '24'.

Figure 10: Scenario Constraints tab - Max shift size and hours per day

#### 4.2.4. Resource Constraints page

This page contains the most important parameters for the optimisation. The resource constraints page allows the end user to customise every resource to

their specifications. You can select a resource from the list of participating resources, as seen in Figure 11. Once a resource is selected three different parameter sections can be edited, illustrated in Figure 12, 13, and 14. Figure 12 illustrates the resource-specific constraints concerning the working

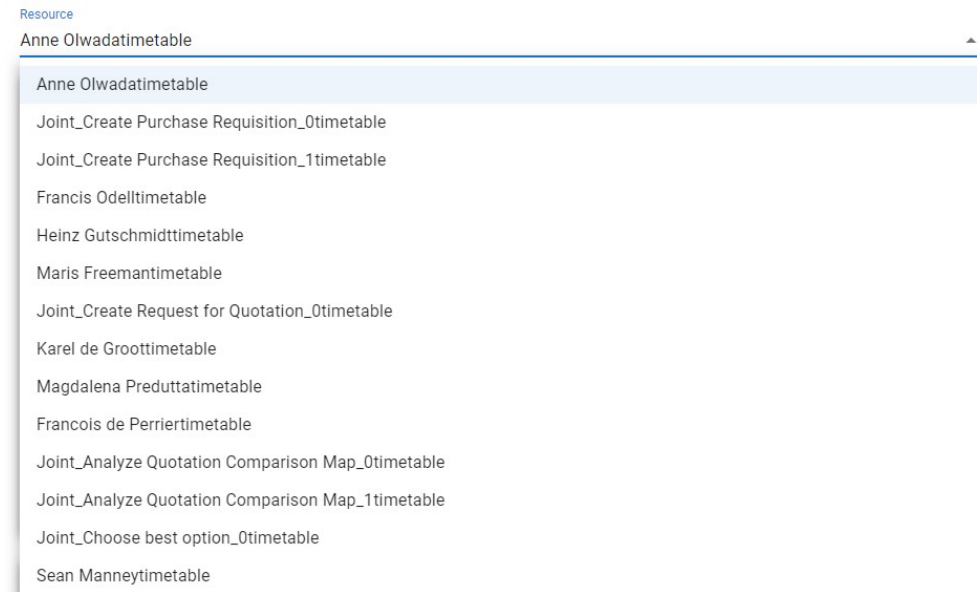


Figure 11: Resource Constraints tab - Resource selection list

parameters. These are:

- Maximum weekly capacity - How many slots the resource can be assigned maximally per week.
- Maximum daily capacity - How many slots the resource can be assigned maximally per day.
- Maximum consecutive capacity - How many slots the resource can perform consecutively before at least one slot is not assigned. A.k.a., the maximum length of a Shift.
- Maximum shifts per day - How many shifts can be assigned to the resource per day. E.g., 2 = The resource can maximally have two shifts assigned per day.

- Maximum shifts per week - How many shifts can be assigned to the resource per week. E.g., 7 = The resource can maximally have 7 shifts assigned per week.
- Human Resource? - Is the resource a human being? This parameter defines whether or not the resource will be optimised or not, since systems are expected to have a near 100% uptime during the times they are active and as such cannot be optimised further.

Resource  
Anne Olwadatimetable

---

**Resource constraints**

Resource max weekly capacity  
125

---

Resource max daily capacity  
19

---

Resource max consecutive capacity  
11

---

Resource max shifts per day  
24

---

Resource max shifts per week  
125

---

☒ Human resource?

Figure 12: Resource Constraints tab - Resource constraints

The next section of this page is illustrated in Figure 13. Here the user can enter a number that represents a binary value corresponding to the shift slots that the resource is not allowed to work in, with a "1" meaning the resource is not allowed to work and a "0" meaning the resource can work during that slot. E.g., 1111 0000 1111 means that the resource cannot work for the first four slots, is then allowed to work in the next four slots, and finally is not allowed to work in the last four slots again.

The final section of this page is illustrated in Figure 14. Here the user can enter a number that represents a binary value corresponding to the shift slots that the resource must work in, with a "1" meaning the resource is must work and a "0" meaning the resource can work during that slot. E.g., 1111 0000 0000 means that the resource must work for the first four slots, and is then allowed to work during the rest of the slots.

Never work masks

|           |   |
|-----------|---|
| Monday    | 0 |
| Tuesday   | 0 |
| Wednesday | 0 |
| Thursday  | 0 |
| Friday    | 0 |
| Saturday  | 0 |
| Sunday    | 0 |

Figure 13: Resource Constraints tab - Never work masks

It is important to note that the *Never Work Masks* and the *Always Work Masks* cannot intersect. Doing so creates a conflict as a resource must work during a slot it is not allowed to work.

Always work masks

|           |   |
|-----------|---|
| Monday    | 0 |
| Tuesday   | 0 |
| Wednesday | 0 |
| Thursday  | 0 |
| Friday    | 0 |
| Saturday  | 0 |
| Sunday    | 0 |

Figure 14: Resource Constraints tab - Always work masks

After all of the parameters have been customised and verified by the user, they can press the *Start Optimisation* button. This action will perform a last check to ensure all parameters are valid and no conflicting masks have

been assigned.

#### 4.2.5. Simulation Results page

On this page, the user will see the results of the optimisation task. This page is illustrated in Figure 15. On this page, the user can see a high-level overview of the optimisation metrics of the performed task including:

- The average cost of the optimal solutions
- The average cycle time of the optimal solutions
- The number of optimal solutions found - Pareto size
- The number of optimal solutions of the chosen approach that are present in the Pareto front. This value is only significant when multiple approaches have been performed.
- Cost optimisation metric. A number representing the average cost optimisation compared to the original model. If the number is larger than one, then the found solutions have improved the average cost of the model.
- Time optimisation metric. A number representing the average time optimisation compared to the original model. If the number is larger than one, then the found solutions have improved the average cycle time of the model.

Below these values, the user can also see every Pareto entry, their identification hash code and the median cost and cycle time of the solution. They can also download the simulation parameter and constraint parameters for each solution.

For a more in-depth look, the user can download all of the related files to the optimisation report by clicking the ***Download entire report*** button. This will create a .zip file, containing all of the found solutions, their parameters and the comparative metric file.

Optimization report

DOWNLOAD ENTIRE REPORT

| Hill climb FLEX - Combined |  |
|----------------------------|--|
| Average cost               | 106178.98527229627                       |
| Average cycle time (sec)   | 115591.94259246916                       |
| Pareto size                | 5  |
| # in Pareto                | 5  |
| Cost compared to original  | 1.0707128894833042                       |
| Time compared to original  | 0.9977731965522868                       |
| Pareto values              |  |
| Entry ID                   | 5b820c2a31ac268a8c2b2779c4f47a3ed9578a25 |
| Median cost                | 106508.33338741334                       |
| Median cycle time (sec)    | 112708.69865688922                       |
| DOWNLOAD PARAMETERS        | DOWNLOAD CONSTRAINTS                     |
| Entry ID                   | 5115fe05c9915dfe71b49ee528c683791f86cced |
| Median cost                | 98919.51871306666                        |
| Median cycle time (sec)    | 117310.43950939097                       |
| DOWNLOAD PARAMETERS        | DOWNLOAD CONSTRAINTS                     |
| Entry ID                   | b0b74d5a8c823b60db8f4e76db7f32e141d222bf |
| Median cost                | 98854.33300796666                        |
| Median cycle time (sec)    | 119227.89739422908                       |
| DOWNLOAD PARAMETERS        | DOWNLOAD CONSTRAINTS                     |
| Entry ID                   | 76fd19fb06ce0c1982634ab354b0cf622c2c7ee2 |
| Median cost                | 121670.54771762498                       |
| Median cycle time (sec)    | 111983.65692795823                       |
| DOWNLOAD PARAMETERS        | DOWNLOAD CONSTRAINTS                     |

Figure 15: Simulation Results tab

## 5. Implementation and Evaluation

The simulation engine is implemented in an open-source simulation engine, namely PROSIMOS, available at <https://github.com/AutomatedProcessImprovement/Prosimos>. PROSIMOS supports the simulation of processes with an unpooled allocation model and differentiated availability and performance. PROSIMOS takes as input a BPMN process model with simulation parameters as per Def. 1 (encoded in JSON format). Like other simulation engines, PROSIMOS produces an event log and a set of performance indicators such as waiting, processing, cycle times, and resource utilisation. For a more detailed explanation of the functionality and architecture of PROSIMOS, we refer the readers to [17].

The proposed simulation-based optimisation approach is implemented as an open-source (Python-based) library, namely OPTIMOS available at <https://github.com/AutomatedProcessImprovement/roptimus-prime>. Like PROSIMOS, OPTIMOS takes a BPMN process model and the simulation parameters in JSON format as input. From those, it produces a set of resource profiles corresponding to the Pareto-optimal resource configurations.

Similarly, using OPTIMOS in conjunction with PROSIMOS, we conducted an empirical evaluation of the simulation-based optimisation approach presented in Section 3. This evaluation aims to answer the following sub-questions derived from the question **RQ1** presented in Section 1: **EQ1** To what extent OPTIMOS improves the cycle time and cost of the models discovered by PROSIMOS? **EQ2** What impact do the five perturbations handled by the optimisation approach have on the discovered Pareto fronts regarding convergence, spread, and distribution?

### 5.1. Datasets

We use five simulated (synthetic) logs and five real-life ones. Since our proposal does not deal with process model discovery, we use the BPMN models generated from the input logs using the Apromore open-source platform,<sup>11</sup>, which we manually adjusted to obtain 90% replay-based fitness. Table 2 gives descriptive statistics of the employed logs, including the number of traces and events and the number of activities and resources. Row “simulation time”

---

<sup>11</sup><https://apromore.com>



|                 | LO-SL/ | LO-SH/ | LO-ML/ | LO-MH/ | P-EX/ | PRD/ | C-DM/ | INS/   | BPI-12/ | BPI-17  |
|-----------------|--------|--------|--------|--------|-------|------|-------|--------|---------|---------|
| Traces          | 1000   | 1000   | 1000   | 1000   | 608   | 225  | 954   | 1182   | 8616    | 30 276  |
| Events          | 9844   | 9782   | 9768   | 9569   | 9119  | 4503 | 4962  | 23 141 | 59 302  | 240 854 |
| Activities      | 15     | 15     | 15     | 15     | 23    | 23   | 18    | 11     | 8       | 9       |
| Resources       | 19     | 19     | 34     | 34     | 47    | 54   | 337   | 125    | 68      | 141     |
| Simulation Time | 1.27   | 1.24   | 1.25   | 1.24   | 1.07  | 0.72 | 0.73  | 1.29   | 10.32   | 41.97   |

Table 2: Characteristics of the business processes used in the experimentation.

shows the average execution times (in seconds) across five simulation runs.

The first four event logs were obtained by simulating a Loan Origination (LO) process model using Apromore. The model contains 15 tasks assigned to 5 resource pools. We first simulated the model by assigning the same calendar to all resource pools. Using this single-calendar (S) model, we generated two logs: one where the resource utilisation of each pool is around 50% (Low Utilization – L) and another with a resource utilisation of 80% (High Utilization – H). The simulation parameters of the H model were identical to the ones of the L model, except that we adjusted the case arrival rate to obtain higher resource utilisation. To test the techniques in the presence of multiple calendars, we simulated the same model after assigning different (overlapping) calendars to each of the five resource pools. We simulated this multi-calendar (M) model twice: once with a low utilisation (L) and once with high utilisation (H). This procedure led to four simulated logs: LO-SL, LO-SH, LO-ML, LO-MH. The fifth log (*purchasing-example (P-EX)*) is part of the academic material of the Fluxicon Disco tool.<sup>12</sup>

The first real-life log (*PRD*) is a log of a manufacturing process.<sup>13</sup> The second and third are anonymized real-life logs from private processes. The *C-DM* comes from an academic recognition process executed at a Colombian University. The *INS* log belongs to an insurance claims process. The fourth real-life log is a subset of the BPIC-2012 log<sup>14</sup> – of a loan application process from a Dutch financial institution. We focused on the subset of this log consisting of activities that have both start and end timestamps. Similarly, we used the equivalent subset of the BPIC-2017 log<sup>15</sup>, which is an updated version of the BPI-2012 log (extracted in 2017 instead of 2012). We extracted

<sup>12</sup><https://fluxicon.com/academic/material/>

<sup>13</sup><https://doi.org/10.4121/uuid:68726926-5ac5-4fab-b873-ee76ea412399>

<sup>14</sup><https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f>

<sup>15</sup><https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b>

the subsets of the BPI-2012 and BPI-2017 logs by following the recommendations provided by the winning teams of the BPIC-2017 challenge.<sup>16</sup>

## 5.2. Experimental setup

To address questions **EQ1-EQ2**, we took as input the simulation models discovered by PROSIMOS. Then, we run OPTIMOS over five different variants defined with respect to the five perturbations in Algorithm 1, as follows:

- **SKD** optimises only the resource calendars, i.e., perturbations 1-3, without adding or removing any resource to the roster (i.e., perturbations 4-5 are not applied).
- **RES** adds or removes resources to the roster using perturbations 4-5 without updating their calendars (i.e., perturbations 1-3 are not applied).
- **SKD  $\cup$  RES** considers all the 5 perturbations. In this variant, OPTIMOS tries to improve the roster at each iteration by adding, removing resources, or updating their calendars according to perturbations 1-5.
- **SKD  $\rightarrow$  RES** first optimises the roster without adding or removing resources, i.e., perturbations 1-3, until the hill-climbing algorithm stops. Next, it optimised the resource allocation by running a second round of the hill-climbing algorithm, this time to optimise the resource allocation using perturbations 4 and 5 (i.e. duplicating or removing existing resources). The second round of hill-climbing starts from the Pareto front obtained in the first round of hill-climbing. In other words, we run **SKD** followed by **RES**.
- **RES  $\rightarrow$  SKD** first optimises the resource allocation by adding or removing resources only, i.e., perturbations 4-5. It then runs a second round of hill-climbing to optimise the roster, i.e., perturbations 1-3, starting the optimal allocation obtained in the first round of hill-climbing. In other words, we run **RES** followed by **SKD**.

---

<sup>16</sup><https://www.win.tue.nl/bpi/doku.php?id=2017:challenge>

In our experiments, in addition to an empty solution candidate queue, we set the maximum number of solutions to explore (i.e., function evaluations) to 5000 and, at most, 500 (10%) consecutive Pareto non-optimal rosters explored as stopping criteria. For all the variants, we run three simulations per roster candidate (using PROSIMOS) to estimate the corresponding times and costs. To avoid giving unfair advantages to any variant due to the stochastic nature of the simulations, we memorised the simulation results in files. So, we can assert that if two variants explore the same roster, they will get identical times and costs. The memorisation reduces the number of simulations, thus the execution times, when multiple variants explore common areas in the solution space. Data about salaries/costs of the resources involved in the process execution is missing in the event logs used by PROSIMOS so in the discovered simulation models. Thus, we assigned each resource with the unitary cost per hour for the experiments.

To answer the experimental question **EQ1**, we computed the mean cycle times and cost of all the Pareto fronts built by each variant. We calculated the improvement ratio by dividing the costs and cycle times of the original solution (discovered by PROSIMOS) and the means of the resulting Pareto front, i.e.,  $initial\_cost_p/mean\_pareto\_cost_p$  and  $initial\_cycle\_t/mean\_pareto\_cycle\_t$ .

In the simulation models used in the experiments, the optimal rosters, i.e., the Pareto front, are unknown. Then, we follow the approach presented in [18, 14], which creates a reference Pareto front  $PRef$  to compare the results retrieved by many solvers. Specifically,  $PRef$  is the set containing the (joint) Pareto-optimal solutions from the entire search space explored by all the runs of the five variants presented above. Henceforth, we will call  $PRef$  the reference Pareto front (joint from all the variants) and  $PApprox$  the approximated (by one variant) Pareto front. Then, to answer the experimental question **EQ2**, we used four metrics:

- Hyperarea [19] ( $HA$ ) measures convergence and distribution. So far, it is considered the most relevant and widely used measure to compare algorithms in the evolutionary community [20]. Hyperarea is the area in the objective space dominated by a Pareto front delimited by a point  $(c, t) \in \mathbb{R}^2$ , which we set as the maximum cost and time among all the solutions explored. If  $PRef$  is available, the hyperarea ratio is a real number between 0 and 1, given by  $HA(PApprox)/HA(PRef)$ . A higher hyperarea ratio means a better  $PApprox$ , being 1.0 the maximum possible ratio indicating that  $PApprox$  dominates the same solution space

as  $PRef$ .

- Averaged Hausdorff distance [21] measures convergence using the distances between  $PAprox$  and  $PRef$ . Specifically, it gets the greatest distance from each point in one set to the closest point in the other set, i.e., given by  $\max(\min\|p_i, PRef\|_2, \min\|p_j, PAprox\|_2)$ ,  $\forall p_i \in PAprox, p_j \in PRef$ . A lower Hausdorff distance means a better  $PAprox$ .
- Delta( $\Delta$ ) [22, 18] measures spread and distribution, given by:

$$\Delta = \frac{d_0 + d_n + \sum_{i=1}^{n-1} |d_i - d'|}{d_0 + d_n + (n-1)d'}$$

where  $d_i, 0 \leq i \leq n = |PAprox|$  is the Euclidean distance between consecutive solutions, with  $d_0$  and  $d_n$  being the Euclidean distances between the extreme solutions in  $PRef$  and the extreme solutions in  $PAprox$ . Besides,  $d'$  is the average of those distances. A lower value of  $\Delta$  means a better  $PAprox$ .

- Purity [18] is a cardinality measure used to compare Pareto fronts built by different algorithms. It is given by  $|PAprox \cap PRef|/|PAprox|$ . Thus, it measures the ratio of solutions in  $PAprox$  included in  $PRef$ . Higher purity means a better  $PAprox$  ratio of non-dominated solutions, with 1.0 being the maximum value possible.

### 5.3. Experimental Results

To answer the question **EQ1**, Table 3 illustrates the ratio between cycle times and cost derived from the initial roster discovered by PROSIMOS, divided by the mean of those in the Pareto fronts obtained by each variant of OPTIMOS. For each variant, the value on the top (in gray) corresponds to the cycle time ratio, and the ones at the bottom are the cost ratios. The results show that the mean values improve in both components, i.e., time and cost, in 37 out of the total 50 evaluations (74%). The remaining evaluations (13 out of 50) show a mean improvement of one component, i.e., 26%. Independently, the variants **SKD**, **SKD**  $\cup$  **RES** show a mean improvement ob both components in 8 out of 10 logs, while **RES**, **SKD**  $\rightarrow$  **RES** and **RES**  $\rightarrow$  **SKD** in 7 out of 10. However, analysing each of the rosters in the Pareto fronts discovered

|                       | LO-SL                                   | LO-SH  | LO-ML  | LO-MH  | P-EX   | PRD    | C-DM    | INS    | BPI-12 | BPI-17 |
|-----------------------|---|--------|--------|--------|--------|--------|---------|--------|--------|--------|
| Mean Time, Cost Ratio | <b>SKD</b>                              | 1.056, | 1.199, | 1.068, | 1.040, | 1.057, | 1.032,  | 1.091, | 1.018, | 1.127, |
|                       |   | 1.073  | 1.090  | 0.996  | 1.056  | 1.068  | 1.045   | 1.130  | 0.999  | 1.096  |
|                       | <b>RES</b>                              | 0.410, | 1.059, | 1.039, | 0.963, | 1.136, | 24.269, | 2.946, | 0.910, | 1.193, |
|                       |   | 1.247  | 1.065  | 1.116  | 1.226  | 1.709  | 1.921   | 3.790  | 1.612  | 2.246  |
|                       | <b>SKD <math>\cup</math> RES</b>        | 0.484, | 1.110, | 1.065, | 1.038, | 1.195, | 34.123, | 2.638, | 0.906, | 1.263, |
|                       |   | 1.225  | 1.099  | 1.106  | 1.193  | 1.759  | 1.781   | 3.297  | 2.449  | 2.314  |
|                       | <b>SKD <math>\rightarrow</math> RES</b> | 0.392, | 1.185, | 1.044, | 0.965  | 1.119, | 25.403, | 2.720, | 1.018, | 1.248, |
|                       |   | 1.243  | 1.100  | 1.115  | 1.238  | 2.118  | 1.917   | 3.321  | 0.999  | 1.884  |
|                       | <b>RES <math>\rightarrow</math> SKD</b> | 0.317  | 1.024, | 1.070, | 0.999, | 1.074, | 24.269, | 2.946, | 0.757, | 1.138, |
|                       |   | 1.356  | 1.120  | 1.096  | 1.252  | 1.863  | 1.921   | 3.790  | 2.760  | 2.349  |

Table 3: Mean cycle times and costs of the Pareto fronts obtained by OPTIMOS.

by each variant, we observed that at least one of the rosters improved the component whose mean value remained below the initial allocation. In other words, all the Pareto fronts contain at least one roster improving cycle time and/or cost.

The results above are sound and aligned with the expectations drawn during the experimental setup. Note that the time and cost dimensions are conflicting, e.g., lower costs due to resource and calendar reductions typically lead to higher cycle times due to lower resource availability and higher resource contention, and vice-versa. Besides, the magnitude of the improvement is restricted by the optimisation constraint. For example, the experiments limited the maximum number of working hours per day/week to the maximum observed in the initial roster availability. In some simulated event logs, those (observed) values were set to 8 hours per day and 40 per week, making the operations performed by OPTIMOS restrictive and closer to the optimal times for the existing resources, i.e., impacting the calendar optimisation. Still, the Pareto fronts in the current results give process managers a broad spectrum to select which roster is more convenient according to their needs. So they can pick an allocation with the lowest cost possible to the detriment of the time. Conversely, they can select a roster with the most downtime and the corresponding cost impact. Or, they can aim for a more balanced time-cost ratio by choosing a resource allocation in the center of the Pareto front.

In multi-objective optimisation, measuring the quality of a Pareto front approximation retrieved by an algorithm is not trivial. According to [2], a good approximation must minimise the distance to the actual Pareto front (a.k.a. convergence). Besides, a good Pareto front should consist of a highly diversified set of points, which are well distributed across the front (a.k.a. spread and distribution). Accordingly, to answer the question **EQ2**, Table 4

|                   |   | LO-SL   | LO-SH   | LO-ML   | LO-MH   | P-EX    | PRD       | C-DM    | INS     | BPI-12 | BPI-17 |
|-------------------|---|---------|---------|---------|---------|---------|-----------|---------|---------|--------|--------|
| Hyperarea         | <b>SKD</b>                              | 0.576   | 0.962   | 0.771   | 0.795   | 0.361   | 0.470     | 0.230   | 0.339   | 0.701  | 0.878  |
|                   | <b>RES</b>                              | 0.942   | 0.947   | 0.973   | 0.915   | 0.858   | 0.984     | 0.959   | 0.965   | 0.989  | 0.956  |
|                   | <b>SKD <math>\cup</math> RES</b>        | 0.884   | 0.999   | 0.949   | 0.978   | 0.884   | 0.981     | 0.998   | 0.988   | 0.999  | 0.999  |
|                   | <b>SKD <math>\rightarrow</math> RES</b> | 0.919   | 0.962   | 0.946   | 0.978   | 0.999   | 0.999     | 0.989   | 0.339   | 0.976  | 0.999  |
|                   | <b>RES <math>\rightarrow</math> SKD</b> | 0.999   | 0.967   | 0.998   | 0.924   | 0.904   | 0.984     | 0.958   | 0.999   | 0.996  | 0.961  |
| Hausdorff         | <b>SKD</b>                              | 90270.7 | 6206.5  | 55127.5 | 94534.1 | 52129.0 | 3249328.0 | 29221.4 | 70639.8 | 2620.6 | 3624.6 |
|                   | <b>RES</b>                              | 8107.8  | 12543.4 | 11104.1 | 40446.7 | 11064.6 | 6831.3    | 3344.8  | 33415.6 | 980.6  | 1965.4 |
|                   | <b>SKD <math>\cup</math> RES</b>        | 8045.7  | 685.8   | 20026.1 | 36170.9 | 7925.2  | 50726.3   | 576.2   | 13037.9 | 123.1  | 235.0  |
|                   | <b>SKD <math>\rightarrow</math> RES</b> | 9402.6  | 6220.2  | 22110.8 | 22482.6 | 3113.4  | 4207.6    | 781.5   | 70639.8 | 842.2  | 236.0  |
|                   | <b>RES <math>\rightarrow</math> SKD</b> | 775.04  | 5068.0  | 6008.8  | 50242.3 | 10690.2 | 6831.3    | 3344.8  | 4243.0  | 675.5  | 2448.1 |
| Delta( $\Delta$ ) | <b>SKD</b>                              | 0.986   | 0.765   | 0.932   | 0.916   | 0.942   | 0.977     | 0.952   | 0.924   | 0.924  | 0.891  |
|                   | <b>RES</b>                              | 1.173   | 0.856   | 0.684   | 0.777   | 0.721   | 0.681     | 1.0     | 0.733   | 0.849  | 0.895  |
|                   | <b>SKD <math>\cup</math> RES</b>        | 1.198   | 0.772   | 0.775   | 0.744   | 0.988   | 0.874     | 0.524   | 1.066   | 0.675  | 0.803  |
|                   | <b>SKD <math>\rightarrow</math> RES</b> | 1.074   | 0.765   | 0.853   | 0.695   | 0.832   | 0.838     | 0.702   | 0.924   | 0.831  | 0.643  |
|                   | <b>RES <math>\rightarrow</math> SKD</b> | 1.076   | 0.795   | 0.778   | 0.814   | 0.937   | 0.681     | 1.0     | 0.667   | 0.73   | 0.920  |
| Purity            | <b>SKD</b>                              | 0.333   | 0.471   | 0.0     | 0.0     | 0.0     | 0.0       | 0.0     | 0.0     | 0.0    | 0.0    |
|                   | <b>RES</b>                              | 0.0     | 0.0     | 0.063   | 0.0     | 0.0     | 0.0       | 1.0     | 0.0     | 0.143  | 0.0    |
|                   | <b>SKD <math>\cup</math> RES</b>        | 0.343   | 0.75    | 0.12    | 0.375   | 0.037   | 0.75      | 0.615   | 0.9     | 0.764  | 0.777  |
|                   | <b>SKD <math>\rightarrow</math> RES</b> | 0.136   | 0.5     | 0.167   | 0.167   | 1.0     | 0.409     | 0.4     | 0.0     | 0.1    | 0.333  |
|                   | <b>RES <math>\rightarrow</math> SKD</b> | 0.659   | 0.214   | 0.9     | 0.8     | 0.045   | 0.0       | 1.0     | 0.417   | 0.231  | 0.0    |

Table 4: Results of the Hyperarea, Hausdorff, Delta, and Purity metrics.

shows the results of the performance metrics achieved by the five variants of Algorithm 1, highlighting the best score for each metric on each of the logs.

Regarding convergence and distribution, the values retrieved by the hyperarea illustrated that the variants spanning all five perturbations achieve better results. The variant **SKD  $\cup$  RES**, which tries all the perturbations at each iteration, scored the best results in 5 out of 10 event logs. Besides, in all the experiments, it achieved hyperarea ratios superior to 0.88 (1.0 is the max possible), meaning that it dominates at least 88% of the solution space dominated by the reference Pareto front. The variant **RES  $\rightarrow$  SKD**, which optimises only the number of resources and then optimises only the calendars of the resulting resources, exhibited the top scores in 3 even logs. Besides, it achieved hyperarea values superior to 0.90 in all the perturbations. Also, regarding convergence, the Hausdorff distance points out the variant **RES  $\cup$  SKD** as the best one, achieving the top values in 4 event logs, the second score in another 4, and the best results overall.

The Delta spread metric did not evidence a bad performance of any variant compared to the others. Overall, it has the most similar results when comparing the variants, with each scoring 2 or 3 of the top scores. A lower value of Delta means a better spread, and in 7 logs, all the variants scored values below 1, while the remaining values were all around 1, which can be considered good results. Finally, the purity rates again highlight the variants **SKD  $\cup$  RES** and **RES  $\rightarrow$  SKD** as the ones adding more solutions to the

joint Pareto fronts, with tops scores in 5 and 4 logs, respectively. Conversely, applying perturbations 1-3 or 4-5 of Algorithm 1 isolated leads to solutions that add no rosters to the joint Pareto front, i.e., variant **SKD** fails to add any allocation in 8 logs, and **RES** in 7. The latter is expected as the combined variants refine the results obtained by the variant **SKD** applying **RES** after, and vice-versa.

To summarise, the experiments show that an optimisation approach combining all the perturbations described by Algorithm 1 achieves better results. Specifically, the evaluation suggests as the best strategy is to iteratively add or remove possibly conflicting resources, or update their calendar, i.e., **SKD**  $\cup$  **RES**, keeping (at each iteration) the rosters that reduce waiting time, idle time, and/or cost, discarding the remaining ones. Alternatively, performing a complete optimisation adding or removing resources (only perturbations 4-5), then optimising the calendars of the resulting rosters (only perturbations 1-3), or vice-versa, i.e., variants **RES**  $\rightarrow$  **SKD** or **SKD**  $\rightarrow$  **RES** respectively, achieves comparable results.

Finally, although the variants **SKD** and **RES** performed inferiorly compared to the combined ones, they still offer good approximations, which are helpful in some scenarios. Optimising only the resource calendar (**SKD**) fits the situations in which organisations have a fixed number of resources, e.g., due to cost or capacity restrictions. Still, they can re-schedule them according to the workload needs, restricted according to the organisational rules, e.g., the maximum number of daily/weekly working hours per resource, on-duty and off-duty periods, etc. Conversely, updating the resources calendars may be unattainable in some situations, e.g., all are humans and already working eight hours and five days a week, while the possible intervals to re-schedule are those in which the organisation is closed. In those cases, adding and removing resources is an alternative to reduce cycle time and cost inefficiencies, e.g. by perturbing the calendars of resources with high or low utilisation.

*Threats to validity.* The evaluation reported above is potentially affected by the following threats to validity: (1) **Internal validity**: the experiments rely only on ten events logs. The results could differ on other datasets. To mitigate this limitation, we selected logs with different sizes and characteristics and from different domains. (2) **Ecological validity**: the evaluation compares the simulation results against the original log. While this allows us to measure how well the simulation models replicate the as-is process, it does not allow us to assess the accuracy improvements of using differentiated

resources in a what-if setting, i.e., predicting the performance of the process after a change.

*Web-app tests.* In the scope of this thesis, we wanted to focus on the performance of the optimisation algorithm. The development of the web application was decided as a means of facilitating the interaction with the optimisation algorithm. The web interface allows less technical users to interact with the algorithm as well. The current implementation of the web interface is not a production-quality interface and thus does not have relevant test results to show. However, we did perform in-house demos, showing the capabilities of the interface, with the goal of receiving feedback for further improvements. The interface is still actively in development and is part of a demo paper that is in the works at the moment.



## 6. Conclusion and future work

This thesis presented an optimisation approach to compute a set of Pareto-optimal resource configurations, minimising resource cost and cycle time, in the setting where each resource has its own availability calendar.

The experiments show how an iterative optimisation approach that adds, removes resources and perturbs their availability calendars (i.e. the roster) at each iteration, leads to a set of Pareto-optimal rosters superior to alternative approaches that optimise the resource allocation and the rosters separately, one after the other.

We also implemented a web interface that facilitates the user experience to interact with the developed application.

The research presented has a few limitations that warrant further research. The optimisation approach relies on a hill-climbing strategy, which exploits a local search of the solution space; thus, it may stop on a local optimal. Another future work direction is to use other searching strategies like tabu-search, and genetic algorithms, which span a more comprehensive range of solutions to explore.

The front-end interface of the application should be addressed as well. Currently, the application is operational but requires a rework of certain visual features. E.g., The current implementation of the working masks makes it difficult for the end user as they would need to perform integer-to-binary conversion to enter the correct mask they wish to use.

**Reproducibility.** The experiments on public datasets may be reproduced by cloning the repository <https://github.com/AutomatedProcessImprovement/roptimus-prime> for the optimisation approach. The repository contains instructions to reproduce the experiments.

## Acknowledgement

While writing this thesis, a digital writing assistant, Grammarly<sup>17</sup>, was used. Grammarly is an AI-powered digital writing assistant that helps users improve their writing by providing suggestions for grammar, spelling, punctuation, clarity, and style. It is available as a web-based service, a browser extension, and a desktop application for Windows and Mac. Grammarly can check various writing types, including emails, essays, reports, etc. It uses advanced algorithms and natural language processing to analyse text and provide suggestions in real-time, allowing users to make corrections and improve the quality of their writing as they go. Some of Grammarly's key features include:

- Grammar and spelling checks
- Punctuation suggestions
- Clarity and readability analysis
- Tone and style suggestions
- Vocabulary enhancement

Grammarly also offers a premium version with additional features such as plagiarism detection, advanced grammar checks, and a broader range of style suggestions.

---

<sup>17</sup><https://app.grammarly.com/>

## References

- [1] M. Larry, Business process optimization: combining project management and six sigma best practices to better understand and optimize critical business processes.
- [2] C. Audet, J. Bignon, D. Cartier, S. Le Digabel, L. Salomon, Performance indicators in multiobjective optimization, *Eur. J. Oper. Res.* 292 (2021) 397–422.
- [3] I. Boussaïd, J. Lepagnot, P. Siarry, A survey on optimization meta-heuristics, *Inf. Sci.* 237 (2013) 82–117.
- [4] P. Senkul, I. H. Toroslu, An architecture for workflow scheduling under resource allocation constraints, *Inf. Syst.* 30 (2005) 399–422.
- [5] S. P. F. Peters, Analysis and Optimization of Resources in Business Processes, PhD dissertation, Technische Universiteit Eindhoven, 2021.
- [6] Y. Si, V. Chan, M. Dumas, D. Zhang, A petri nets based generic genetic algorithm framework for resource optimization in business processes, *Simul. Model. Pract. Theory* 86 (2018) 72–101.
- [7] Y. Yu, M. Pan, X. Li, H. Jiang, Tabu search heuristics for workflow resource allocation simulation optimization, *Concurr. Comput. Pract. Exp.* 23 (2011) 2020–2033.
- [8] Z. Huang, X. Lu, H. Duan, A task operation model for resource allocation optimization in business process management, *IEEE Trans. Syst. Man Cybern. Part A* 42 (2012) 1256–1270.
- [9] H. Lee, S.-S. Kim, Integration of process planning and scheduling using simulation based genetic algorithms, *Int. J. Adv. Manuf. Technol.* 18 (2001) 586–590.
- [10] A. Djedović, E. Žunić, Z. Avdagić, A. Karabegović, Optimization of business processes by automatic reallocation of resources using the genetic algorithm, in: *IEEE BIHTEL 2016 Proceedings*, pp. 1–7.
- [11] F. Durán, C. Rocha, G. Salaün, Analysis of resource allocation of BPMN processes, in: *ICSOC 2019 Proceedings*, pp. 452–457.

- [12] J. Xu, C. Liu, X. Zhao, S. Yongchareon, Z. Ding, Resource management for business process scheduling in the presence of availability constraints, *ACM Trans. Management Inf. Syst.* 7 (2016) 9:1–9:26.
- [13] Z. Huang, W. van der Aalst, X. Lu, H. Duan, Reinforcement learning based resource allocation in business process management, *Data Knowl. Eng.* 70 (2011) 127–145.
- [14] O. López-Pintado, M. Dumas, M. Yerokhin, F. M. Maggi, Silhouetting the cost-time front: Multi-objective resource optimization in business processes, in: *BPM Forum 2021*, Springer, 2021, pp. 92–108.
- [15] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, *Fundamentals of Business Process Management*, Second Edition, Springer, 2018.
- [16] T. Weise, *Global optimization algorithms-theory and application*, Self-Published Thomas Weise (2009).
- [17] O. López-Pintado, I. Halenok, M. Dumas, Prosimos: Discovering and simulating business processes with differentiated resources, in: *EDOC 2022 - Demo Track*.
- [18] A. L. Custódio, J. F. A. Madeira, A. I. F. Vaz, L. N. Vicente, Direct multisearch for multiobjective optimization, *SIAM J. Optim.* 21 (2011) 1109–1140.
- [19] J. Wu, S. Azarm, Metrics for quality assessment of a multiobjective design optimization solution set, *Journal of Mechanical Design* 123 (2001) 18–25.
- [20] C. Audet, J. Bigeon, D. Cartier, S. Le Digabel, L. Salomon, Performance indicators in multiobjective optimization, *Eur. J. Oper. Res.* 292 (2021) 397–422.
- [21] O. Schütze, X. Esquivel, A. Lara, C. A. C. Coello, Using the averaged hausdorff distance as a performance measure in evolutionary multiobjective optimization, *IEEE Trans. Evol. Comput.* 16 (2012) 504–522.
- [22] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Trans. Evol. Comput.* 6 (2002) 182–197.

# Appendix

## I.I Dockerfile - presentation tier

```
FROM node:alpine as build

WORKDIR /app
COPY package.json package.json
COPY package-lock.json package-lock.json
COPY .env.production .env.production
COPY vite.config.ts vite.config.ts
COPY index.html index.html
RUN npm ci

COPY tsconfig.json tsconfig.json
COPY tsconfig.node.json tsconfig.node.json
COPY ./public/ ./public
COPY ./src/ ./src
COPY ./nginx/ ./nginx
RUN npm run build

FROM nginx:stable-alpine as production
COPY —from=build /app/dist /usr/share/nginx/html

COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
EXPOSE 443
CMD ["nginx", "-g", "daemon off;"]
```

## I.II Docker-compose configuration - application tier

```
version: "3.7"
services:
  rabbit:
    hostname: rabbit
    image: rabbitmq:3-management
    container_name: optimos-service-rabbitmq
    ports:
      - 5672:5672 # port for the worker for tasks management
      - 15672:15672 # GUI port
  worker:
    build:
      context: .
      dockerfile: Dockerfile.worker
    container_name: optimos-service-celery-worker
    environment:
      - CELERY_BROKER_URL=amqp://rabbit:5672/
      - CELERY_RESULT_BACKEND=redis://redis:6379/
      - FLASK_DEBUG=1
    volumes:
      - ./src/celery/data:/app/src/celery/data
    depends_on:
      - rabbit
      - redis
  api:
    build:
      context: .
      dockerfile: Dockerfile.api
    container_name: optimos-service-api
    ports:
      - 5000:5000
    volumes:
      - ./src/celery/data:/app/src/celery/data
    environment:
      - CELERY_BROKER_URL=amqp://rabbit:5672/
      - CELERY_RESULT_BACKEND=redis://redis:6379/
      - FLASK_DEBUG=1
  redis:
    image: redis:6-alpine
    ports:
      - 6379:6379
```

## II. Licence

### Non-exclusive licence to reproduce thesis and make thesis public

I, **Jonas Berx**,  
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to  
reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,  
**Optimisation of Business Processes with Differentiated Resources**,  
(title of thesis)  
supervised by Marlon Dumas and Orlenys López-Pintado.  
(supervisor's name)
2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Jonas Berx  
**09/05/2023**