Adaptive Storage Rearrangement and Process Discovery in a Warehouse

Christian Stephansen
Aalborg University
DEIS1011F18
csteph13@student.aau.dk

Gideon J. B. Blegmand
Aalborg University
DEIS1011F18
gblegm13@student.aau.dk

June 8, 2018

Abstract

In this thesis we present two parts: Adaptive storage rearrangement, and process discovery, where the second part adds to the work we did in the previous semester. The work for this thesis has been carried out in collaboration with two stakeholders, Av Form which is a warehouse located in Denmark, and Logimatic whom develop the warehouse management system called LOGIA, which is being used at Av Form.

In the First part of the thesis we present the warehouse and how part of their work is structured, followed by the specification and clarification of an area which a worker at the warehouse believes could be improved. We come to the conclusion that the area suited for optimisation relates to the process of order picking, which is believed by many to be the most expensive part of running a warehouse.

We proceed to explain how work done through LOGIA affect the configuration of the warehouse, i.e. where items are stored, their quantity etc., and how this work is being logged in LOGIA. We then present our theory which on the basis of the LOGIA log, is used to devise a novel algorithm that quantifies the impact of rearranging storage locations for a given warehouse configuration. Using this algorithm it is possible to predict which rearrangements in the current warehouse configuration will result in reduced costs for the warehouse.

We then include time heuristics of specific operations used as part of the prediction, before we introduce our implementation of the rearrangement algorithm in the LOGIA system—an implementation which have been used by Av Form.

The LOGIA implementation concludes the first part of our thesis, before we proceed to the second part regarding process discovery. Based on work we conducted during our previous semester, where we amongst other work presented a method for the discovery of timed-arc Petri nets, we elaborate on the time dilemma which emerges during the discovery process. To solve this issue we propose two methods which preprocess a timed event log before it is used in the discovery process.

After our elaboration of the time dilemma and its solutions, we re-introduce our method for calculating the distance between an extended timed-arc work net and a discrete-time event log, where we include an example based on the context of our current stakeholders.

Our thesis then culminates in four experiments covering different areas of the work we have conducted. The first experiment serves as an evaluation of our rearrangement algorithm, the second experiment is used to evaluate the solutions to the time dilemma, our third experiment covers our data mining and process discovery efforts, where we use the solution for the time dilemma chosen in the previous experiment. In our fourth and last experiment, we see how our distance measure evaluates on the discovered models and discrete-time event logs from Av Form.

And finally we conclude on the entire thesis and present suggestions and thoughts on areas that can be researched further in the future.

Keywords: Process mining, Extended timed-arc workflow nets, Distance measure, Partitional clustering, Warehouse management, Storage assignment problem, Adaptive storage relocation

Acknowledgements

The thesis contains information related to both our stakeholders, and therefore we have sought acknowledgement from both parties, that we are allowed to make the content of the thesis public, and that they agree with the statements made throughout. We have received acknowledgement from Bjarke Christensen and Karsten Bangshaab at Logimatic, as well as John Ostersen at Av Form. We appreciate the willingness of these individuals, as well as both Logimatic and Av Form, to engage in this project providing us with a real world setting for our contributions—without them this project would not have been as purposeful.

1. Introduction

In order to reduce costs for a business, and to keep up with competitors, workflow modelling and analysis is often seen as a facilitator [1, 2]. In this thesis we involve one of two stakeholders, a warehouse with whom we identify an area suitable for optimisation. The area we look into is a segment of the warehouse's order picking process, which a number of papers claim to be the most expensive process for a warehouse [3, 4, 5, 6].

We also include another stakeholder, the developer of the Warehouse Management System (WMS) running at the warehouse, and with the two stakeholders we identify how different actions in the WMS affects the configuration of the warehouse. We then use this knowledge to create the theory for suggesting changes to the warehouse configuration, which over time would result in less resources spend on the segment of the order picking process.

The shift from analog to digital technologies has benefitted the world in several regards [7]. One benefit of this technological shift is the capability to store a larger amount of information. According to [7] this has resulted in a compound annual growth rate in storage capacity of 23 % in the period 1986–2007.

These annual increases allow organisations, such as businesses or government departments, to expand and extend on what information is stored. This advancement is encompassed in the growing importance of the concepts of big data and data-driven decision making, which both are intrinsic to the field of data science. Organisations that are capable of exploiting data science, that is to turn data into value, can substantially improve organisational performance [8].

However, exclusively focusing on a data-driven perspective may encumber the potential performance improvements. Organisations are complex structures with even more complex management structures. In order to effectively improve the performance of an organisation this process-centric perspective must also be taken into consideration.

Aalst [9] introduces *process mining* as a means to "*bridge the gap between data science and process science*", where process mining aims at discovering, monitoring and improving real processes based on knowledge obtained from the underlying data referred to as event logs. These event logs can be used to conduct three types of process mining: *Discovery*, *conformance*, and *enhancement*. In the first type, discovery, a process model is created from the event logs without using any a-priori information. The second type, conformance, is utilised to compare a process model to an event log of the same process. The purpose is to ensure that reality, that is the event log, conforms to the model and vice versa. In the last type, enhancement, the aim is to improve or extend a process model using its event logs.

In a paper by Aalst et al. [10], the authors mention three process mining perspectives; process, organisational, and case. We focus mainly on the process perspective for a process mining performed on a use case in this thesis, as we have a focus on the ordering of actions. Where we include work we have done in a provious project (see [11]), where we extended this notion as we included a focus on timed delays between the actions.

Through our previous work with the warehouse and developers of the WMS, we found correlation with findings in [3, 4, 5, 6], that the picking task for a warehouse is the most expensive task in the warehouse, therefore in correlation with two stakeholders, we look into the processes at the warehouse to determine and recommend possible changes to optimise on their overall processes associated with picking.

In the presented work we also apply our previous work on process mining discovery and conformance checking onto workflows mined from event logs from the warehouse, with a focus on conformance checking

different workflows associated with different workers.

Bibliographical Remarks

Some of the work presented in this thesis is an extension to work we did in a previous semester—see [11]. Consequently, some of the sections in this thesis are repeated from [11], including large parts of the above thesis introduction. Also the description of Av Form in Section 5.1, and LOGIA and the associated LOGIA log in Section 5.3. Next is Section 10 covering preliminary knowledge used in Part II. Finally we also included Section 12, with the exception of Section 12.2, as this describes the method we developed for calculating the distance between a extended timed-arc workflow net and a discrete-time event log. The definition of a discrete-time event log and an action in Section 6.2, plus parts of the accompanying text, is taken from our previous work but supplemented with examples from our thesis context.

2. Related Work

Throughout our research, we have found that there is much research on predicting an optimal assignment on storage onto static locations in warehouses [12, 3, 13, 4, 14]. The drawback of the papers is that they only calculate static storage assignment once. Unlike Kofler et al. [5] and Tsamis et al. [6] where the authors research adaptive storage location. However, the research done into this field is sparse.

In their definition of the *storage location assignment problem*, Gu et al. [15] research the allocation of storage locations throughout the lifetime of a warehouse. However, this allocation is still only performed as new items arrive at the warehouse and as a consequence this only concerns the initial assignment of storage locations, whereas our solution looks into relocating items after their initial storage location assignment.

In our process mining, we utilise ProM to mine a petri net using inductive minor, which has been implemented as a plugin into ProM, and later expanded upon [16, 17]. In our work we do not contribute with an extension of the tool, though we use it in an extended fashion, as we perform preprocessing of the event logs used in ProM, in order to combine the generated nets with timed information producing extended timed-arc workflow nets.

3. Our Contribution

In [15] the authors present an overview of the problem of storage assignment of items in a warehouse. In this thesis we devise a novel algorithm that considers the storage assignment problem as a continuous process where the storage assignment must adapt to the ever changing customer demands (e.g. for seasonal or trending items). We focus on developing an algorithm such that it is optimised for the relocation of items as opposed to the initial assignment of items to locations. With this focus in mind, and with the assumption that the storage locations of a warehouse are being utilised, the goal of the algorithm becomes to suggest locations to swap such that storage assignment is optimised, reducing associated costs.

In Figure 1 we provide an illustration of the storage allocation process our proposed algorithm operates in. The swap opportunity represents any moment in which the worker has the time or inclination to perform a swap. We emphasise that we contribute to this step by encapsulation swap suggestions in a box. We only provide the worker with swap suggestions and it is therefore up to the worker to decide, based on their domain knowledge, if a swap is viable; hence, the worker can cancel or execute the swap. The choice is logged in a timed log, and the process repeats itself.

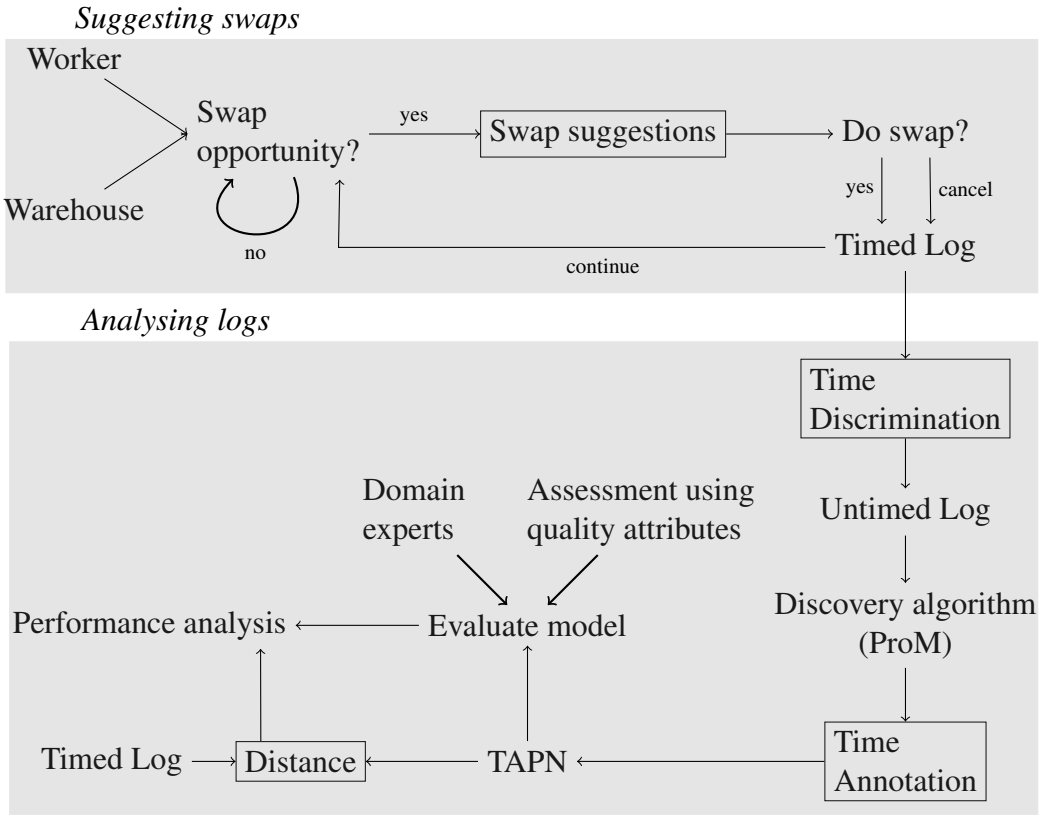


Figure 1: *Illustration of our contribution*

The timed log contains information that is essential to gain insight into the effectiveness of the swap suggestions. In order to analyse the logs and gain this insight we perform process mining [9]. However, before we can effectively process mine the logs we first need to resolve an issue of time bias uncovered as part of a process mining effort presented in our previous work in [11], where we included a method that was generous as it did not include details of time information. Hence, for this thesis we discuss a way to introduce time discrimination of the events in a timed event log such that time bias is avoided. Once the timed log has been pre-processed we can, using the tool ProM [18], discover a model where we do not use the timed information from the log. As we have done in our previous work we shall employ the theory of Timed-Arc Workflow Nets when constructing a model where we combine the discovered time information, and the model generated using ProM. The method of evaluating such a model from a process mining perspective is carried out from a subjective and objective approach. The subjective approach, in the form of domain expert input, provides knowledge of the workflow represented in the model that only a person with intricate knowledge of the workflow can provide. This is then supplemented with the objective approach, in the form of quality attributes [9, pp.188-192], where the characteristics of the model itself are evaluated.

Finally, when an appropriate model has been constructed it can be used for performance analysis. For this purpose we will apply an algorithm, devised in our previous work [11], to find the distance between a timed log and the constructed model. Furthermore, we include a number of ways in which we conduct performance analysis.

4. Outline

In Part I which corresponds to *Suggesting swaps* in Figure 1, we look into the field of adaptive storage rearrangement i.e. swapping storage locations. In addition we in Section 5 present the two stakeholders Av Form, and Logimatic and we present the specific area of the warehouse which is to be improved. In Section 6 we define a warehouse, a warehouse configuration, and a configuration interface formalising how a configuration can change, lastly we define a discrete-time event log. In Section 7 we present the theory and an algorithm to compute swap suggestions in a given warehouse configuration. In Section 8 we estimate the time workers at Av Form use on specific actions, and in Section 9 which concludes the first part, we introduce our implementation of the swap suggestion in to the warehouse management system LOGIA.

In Part II which corresponds to *Analysing logs* in Figure 1, we first introduce work we conducted in a previous semester where we in Section 10 present our label extension to the theory of extended timed-arc Petri nets as well as the theory of extended timed-arc workflow nets (ETAWFNs). In Section 11 we elaborate on the discovery process and what we refer to as the time dilemma with possible solutions related to this time dilemma. And in Section 12 we present a distance measuring algorithm we developed during our previous semester, for calculating the distance between an ETAWFN and discrete-time event log. In Section 13 we conduct four experiments related to our work presented throughout this thesis. We conclude our thesis in Section 14, and lastly in Section 15 we cover suggestions for future work.

Part I
Adaptive Storage Rearrangement

5. Stakeholders

Research in optimisation of warehouses has existed since the 1970's [12]. In [3], the authors give a literature review of work associated with "Design and control of warehouse order picking". A number of papers [3, 4, 5, 6] claim that order picking is the most crucial, expensive, or labour intensive part of a warehouse, where most of the papers [4, 5, 6] claim that more than 50 % of resources (time and/or money) goes to this process alone. From these claims, it naturally follows that any optimisation in the order picking process, would be beneficial for any warehouse as it reduces related costs. As part of this project we collaborate with Av Form [19], as our case-study. In the following we give an overview of this warehouse, followed by a description of the problem we will be looking into.

5.1. Av Form

Av Form is an online webshop selling a broad arsenal of things bought both by private shoppers or e.g. schools. The company primarily has manual labour, and their main office and warehouse is located in Herning in Denmark. The company has existed since 1972, where they originally supplied schools with material for creativity and learning. Our main contact at Av Form is John Ostersen, and he has confirmed that the process of picking is indeed the most expensive process they have. Together with Ostersen, we have isolated an area which is ideal for optimisation, but before presenting this area we introduce some terminology followed by an explanation on how the warehouse, and certain processes are structured at Av Form.
The terminology used in the warehouse, is as follows:

Item A single object, with a unique identifier

Carrier A container that can hold a multiset of items, with a unique identifier

Location Uniquely identifiable place in the warehouse, where carriers are placed

Forward location An identifiable location from which workers can manually collect items

Reserve location An identifiable location placed high, so a forklift is needed to collect the item

Events that occur at Av Form, shortly introduced as:

Pick Removing an item from a carrier in a forward location

Replenish Restocking carriers in forward locations with items from reserve locations

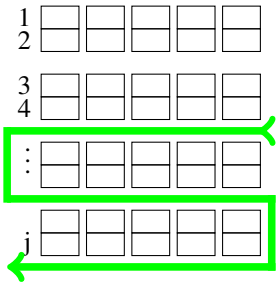


Figure 2: *Sketch of routing path, top view*

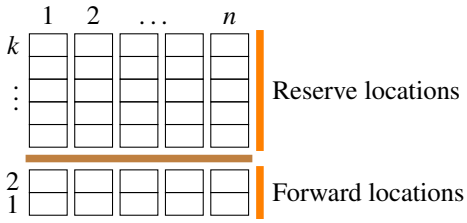


Figure 3: *Sketch of aisle layout, front view, with n racks and k shelves*

The following descriptions are selected for their relevance to Av Form, for more descriptions we refer to Koster et al. [3].

Warehouse Structure The warehouse mainly persists of manual labour, where the warehouse is structured as a *picker-to-parts* [3, p. 483] warehouse, meaning all workers (the pickers) have to move to stationary locations in order to collect items (parts).

Route Planning All routing is planed as an *s-shape routing* [3, p. 495] as illustrated in Figure 2, where workers perform picks on either side, as they traverse down an aisle. If the item furthest away is located at aisle i where $1 \geq i \leq j$ then the worker only has to move to aisle i to initiate the picks, rather than always moving to aisle 1 (the aisle furthest away from the end destination).

Storage Arrangement Finally, the storage locations are arranged such that the warehouse has *forward-reserve allocation* [3, p. 488]. This allocation is depicted in Figure 3 where, similar to the description in [3], workers in the warehouse perform picks directly from the forward area, and use the reserve to replenish items in the forward area.

Figure 3 depicts a situation where all racks in an aisle has the same number of shelves, and the shelves have the same height and width. However, this is not the case at Av Form where the number of shelves, and their height and width may vary. Though the amount of shelves may vary in both forward and reserve areas, the location numbering is always continued across the areas in a rack, i.e. if the top most forward location in a rack is i, then the lowest reserve location in that rack is at $i + 1$.

5.2. The Relocation Problem

We had an initial talk with John Ostersen, to identify problem areas, or areas where improvements would introduce an increase in value for the company. Ostersen had talked with management beforehand, and we discussed both ideas presented by us and by Ostersen, where some ideas were identified by Ostersen as being potential areas where a problem could exist, whereas other ideas were not an area of concern to Ostersen.

We completed the talk by having Ostersen reflect over the different ideas, and tell us which would generate most value for the company. He returned to an original idea proposed by him, which we explain in the following.

The problem presented is in relation to replenishment, which is about moving items from carriers on the reserve locations to carriers on the forward locations. There is a number of things we have to consider:

- We are not to relocate items in the forward locations

- We are to reduce the physical distance travelled when retrieving items for replenishment, considering both floor distance and height to locations
- We must consider that the relocation is also time consuming.

In total we can describe the main problem as:

Is it possible to rearrange reserve locations, such that the total time used for replenishment is reduced?

5.3. LOGIA Warehouse Management and Logistics System

Av Form uses the system called LOGIA [20], to manage their warehouse. LOGIA is both a warehouse management and logistics system created by a department in the company *Logimatic* [21]. The department currently consists of 15 developers whom support, maintain, and expand different versions of the LOGIA system. LOGIA supports both manual and automated work, for Av Form this means mainly support for their manual labour. The system logs most of the work it performs, mostly for the purposes of error handling and support in general. In the following we elaborate on the specific columns and information from the LOGIA log that are relevant to the relocation problem for Av Form — stated in Section 5.2. The relevant columns are as follows:
CARRIER_NO is a unique ID, for every carrier registered in LOGIA.
LOCATION_NO is a unique ID, for every location registered in LOGIA.
ITEM_NO is a unique ID, for every item registered in LOGIA.
TIMESTAMP is used to track when an event is registered in LOGIA.
ACTION is a shorthand for one of the following *action types*:

Following actions are *atomic action types*		
I	In	Place an item into a carrier
O	Out	Inform system that a pick has been performed
C	Count	Manual count of items in a carrier
M	Move	Move entire carrier
Following actions are *logical action types*		
RO	Replenish Out	Remove some quantity of an item from a reserve location, and place onto a temporary carrier
RI	Replenish In	Add item taken from a temporary carrier, onto a forward location

Table 1: *Actions logged in LOGIA*

USER is the person whom executed the task in LOGIA

The action types are divided into atomic action types; that is action types that are irreducible, and logical action types which are inferred from the atomic action types. The action types presented here represents a subset of the action types logged by LOGIA. The relevance of a given action type is based on a distinction between external action types of the environment and internal action types of LOGIA. Since the goal of the relocation problem is to reduce work time we focus on external action types.

6. Theory Overview

In this section we introduce definitions for a warehouse, a configuration and a configuration interface, which covers how a configuration can be changed, using a set of functions. We present a definition for a discrete-time event log, including the definition of an action.

6.1. Warehouse Configuration

A warehouse following Definition 6.1 has many possible states, henceforth referred to as *configurations*. Colloquially, a configuration is the distribution of items in a warehouse to the locations in the warehouse where each location can hold zero or more items. In this section we first formally define the warehouse, followed by the definition of a configuration, and finally we will present the definition of a configuration interface, showing how the action types described in Table 1 affects the configuration.

Definition 6.1 (*WH*)**.** A *Warehouse* (WH) is a quintuple $WH = (L, I, Dist, Lift, Swapable)$ where

- L is a finite set of *locations*,
- I is a finite set of *items*,
- $Dist : L \times L \to \mathbb{N}^0$ is a function assigning the travel time between two locations,
- $Lift : L \to \mathbb{N}^0$ is a function assigning a lift time to a location, and
- $Swapable \subseteq L \times L$ is a symmetric and irreflexive relation stipulating which locations are allowed to be swapped.

Definition 6.2 (*Configuration*)**.** The configuration $C = (Config, Lookup)$ of a warehouse $WH = (L, I, Dist, Lift, Swapable)$ is defined as:

- $Config : L \to (I \to \mathbb{N}^0)$ which is a function that, for every location and item, returns the quantity of an item on a specific location, and
- $Lookup : I \to L^*$ is a function assigning a sequence of locations to items indicating where an item is stored.

In a configuration it is first and foremost required that one can determine the items on a given location and their quantity on said location. It is for this purpose that we define *Config*. The second function *Lookup* is defined for two reasons. First, an item in a warehouse can reside in multiple locations e.g. at Av Form a number of reserve locations will hold the same item as one forward location. Secondly, a warehouse will continuously receive new items and in an effort to be efficient, preempt a surge in demand by ordering additional items. In light of this fact it is important that older low quantity carriers are emptied in order to make room for new ones, avoid item starvation and maximise space utilisation. We therefore stipulate that the sequence of locations returned by *Lookup* is treated as a queue in this regard.

A warehouse can transition from one configuration to another when a worker performs some action type. We refer to the aggregated actions a worker can perform for a given configuration as the *configuration interface*. We denote an interaction from a worker, or a transition as seen from the configuration, as $C \xrightarrow{action} C'$. As a preliminary step we define the notation for renaming elements in a sequence of locations as seen in Definition 6.3.

Definition 6.3 (Renaming)**.** Given a set of elements Σ and a sequence from Σ^*, where $a, b, c \in \Sigma$ and $w \in \Sigma^*$, we define the renaming of all b elements to c as:

$$(a \circ w)\left[{}^{c}/_{b}\right] = \begin{cases} a \circ (w\left[{}^{c}/_{b}\right]) & \text{if } a \neq b \\ c \circ (w\left[{}^{c}/_{b}\right]) & \text{if } a = b \end{cases}$$

$$\epsilon\left[{}^{c}/_{b}\right] = \epsilon$$

We introduce the option of renaming, as a way of preserving age information on items when they are being relocated. In the following we present the definition of the configuration interface, where each function has a

relation to one or more of the action types presented in Table 1. The *In* and *Out* functions map to the I and O actions respectively, whereas the *MoveC* function in accordance with explanations provided by Logimatic, maps to two M actions, where the first action would carry information on items being removed from one location, and the other M action would carry information on items being added to a location. The last function, *MoveI*, we present in the definition has also been specified by Logimatic as consisting of two actions; the RO and RI actions, which has the property of carrying age information on items that are being moved around in the warehouse.

Definition 6.4 (Configuration Interface)**.** For Av Form we define the interface as follows, based on the action types defined in Table 1:

- $C \xrightarrow{In(\bar{l},(\bar{i},\bar{n}))} C'$: In a configuration C a worker performs the action type; I. Consequently, $\bar{n}$ of item $\bar{i}$ are placed into location $\bar{l}$ resulting in a new configuration C' such that

$$precondition : \bar{n} > 0,\ \bar{l} \notin Lookup(\bar{i})$$

$$Config'(l)(i) = \begin{cases} Config(l)(i) & \text{if } l \neq \bar{l} \vee i \neq \bar{i} \\ Config(l)(i) + \bar{n} & \text{if } l = \bar{l} \wedge i = \bar{i} \end{cases} \qquad Lookup'(i) = \begin{cases} Lookup(i) & if\ i \neq \bar{i} \\ Lookup(i) \circ \bar{l} & if\ i = \bar{i} \end{cases}$$

- $C \xrightarrow{Out(\bar{l},(\bar{i},\bar{n}))} C'$: In a configuration C a worker performs the action type; O. Consequently, $\bar{n}$ of item $\bar{i}$ are removed from location $\bar{l}$ resulting in a new configuration C' such that

$$precondition : \bar{n} > 0,\ Lookup(\bar{i}) = \bar{l} \circ tail$$

$$Config'(l)(i) = \begin{cases} Config(l)(i) & \text{if } l \neq \bar{l} \vee i \neq \bar{i} \\ Config(l)(i) - \bar{n} & \text{if } \begin{cases} l = \bar{l} \wedge i = \bar{i}, \\ Config(l)(i) > \bar{n} \end{cases} \\ 0 & \text{if } \begin{cases} l = \bar{l} \wedge i = \bar{i}, \\ Config(l)(i) <= \bar{n} \end{cases} \end{cases} \qquad Lookup'(i) = \begin{cases} Lookup(i) & \text{if } i \neq \bar{i} \\ tail & \text{if } Config'(\bar{l})(\bar{i}) = 0 \wedge i = \bar{i} \\ \bar{l} \circ tail & \text{otherwise} \end{cases}$$

- $C \xrightarrow{MoveC(\bar{l_1},\bar{l_2})} C'$: In a configuration C a worker performs a sequence of two M actions. As a consequence the carrier on location $\bar{l_1}$, with all its contents, is moved to the location $\bar{l_2}$. We formalise this as follows:

$$\begin{aligned} precondition :& \forall i \in I : Config(\bar{l_2})(i) = 0, \\ & \exists i \in I : Config(\bar{l_1})(i) > 0, \bar{l_1} \in Lookup(\bar{i}) \end{aligned}$$

$$Config'(l)(i) = \begin{cases} Config(l)(i) & \text{if } l \neq \bar{l_1} \wedge l \neq \bar{l_2} \\ Config(\bar{l_1})(i) & \text{if } l = \bar{l_2} \\ 0 & \text{if } l = \bar{l_1} \end{cases} \qquad Lookup'(i) = Lookup(i)\left[\bar{l_2}/\bar{l_1}\right]$$

- $C \xrightarrow{MoveI(\bar{l_1},(\bar{i},\bar{n}),\bar{l_2})} C'$: In a configuration C a worker performs the action type sequence of RO followed by RI. As a result hereof; $\bar{n}$ of item $\bar{i}$ are moved from location $\bar{l_1}$ to location $\bar{l_2}$.

$$precondition : \bar{n} > 0, \bar{l_1} \in Lookup(\bar{i})$$

$$Config'(l)(i) = \begin{cases} Config(l)(i) & \text{if } l \neq \bar{l} \vee i \neq \bar{i} \\ Config(l)(i) - \bar{n} & \text{if } \begin{cases} l = \bar{l_1} \wedge i = \bar{i}, \\ Config(l)(i) > \bar{n} \end{cases} \\ 0 & \text{if } \begin{cases} l = \bar{l_1} \wedge i = \bar{i}, \\ Config(l)(i) =< \bar{n} \end{cases} \\ Config(l)(i) + \bar{n} & \text{if } \begin{cases} l = \bar{l_2} \wedge i = \bar{i}, \\ Config(\bar{l_1})(i) > \bar{n} \end{cases} \\ Config(l)(i) + \\ Config(\bar{l_1})(i) & \text{if } \begin{cases} l = \bar{l_2} \wedge i = \bar{i}, \\ Config(l)(i) =< \bar{n} \end{cases} \end{cases} \qquad Lookup'(i) = \begin{cases} Lookup(i) & \text{if } \bar{l_2} \in Lookup(i) \\ Lookup(i) & \\ \quad \left[\bar{l_1} \circ \bar{l_2} / \bar{l_1}\right] & \text{if } \bar{l_2} \notin Lookup(i) \end{cases}$$

6.2. Discrete-Time Event Log

An essential part of any warehouse, or any business for that matter, is to record what transpires on a daily basis. Such information is recorded in event logs and serve as the starting point for what is known as process mining. Process mining is a field that combines a data-driven approach with a process oriented approach in order to analyse the processes of business, organisations, etc. [9]. In the presented work this process analysis is an intrinsic part of our analysis of the practical effectiveness of our swap suggestion algorithm. In this section we shall define what an event log is, and focus on defining the minimum required information present in an event log for it to be used in our analysis efforts. We first define how an action is recorded in an event log as seen in Definition 6.5, where we let $\mathbb{Z}$ be the set of all positive, and non-positive integers including zero.

Definition 6.5 (Action)**.** Given a warehouse $WH = (L, I, Dist, Lift, Swapable)$ then an action is an element $(t, l, i, n, w, o) \in T \times L \times I \times \mathbb{Z} \times W \times O$ where

- T is a finite set of *types*,
- L is a finite set of *locations*,
- I is a finite set of *items*,
- $\mathbb{Z}$ is the quantity difference for an item on a location,
- W is a finite set of *workers*,
- O is a finite set of *order numbers*, and
- the set of all actions is denoted by Act.

Take Table 2 as an example of an action. The elements of the action, except for its type, are only meant to illustrate an action in the context of our case study; hence, the values of the elements are hypothetical. The action type O, described previously in Table 1, indicates that items were removed from the location '123-12-3-A'. In this case the item was taken from the carrier on the location in row 123, rag 12, height 3, position A. From the location, a quantity of five of the item seven were removed by Avery for order 23.

Type	Location	Item	Quantity	Worker	Order number
O	123-12-03-A	7	-5	avr	23

Table 2: *Example of an action which illustrates how actions are logged at Av Form*

An event log is then a sequence of actions where each action is associated with time i.e. an action occurred at some point in time. We define an event log as seen in Definition 6.6 inspired by the definition given by Weijters et al. [22]. In this definition let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

Definition 6.6 (Discrete-time Event Log)**.** A discrete-time event log Π is a multiset of traces π where

- $(a, d) \in Act \times \mathbb{N}_0$ is a *discrete-time event* where
  - Act is a set of actions adhering to Definition 6.5,
- $\pi \in (Act \times \mathbb{N}_0)^+$ is a *discrete-time event trace* such that
  - for a given trace $\pi = \{(a_1, d_1), (a_2, d_2), \ldots, (a_n, d_n)\}$ we have that $d_i \leq d_{i+1}\ \forall i,\ 1 \leq i \leq n$, and
- $\Pi \subseteq (Act \times \mathbb{N}_0)^+$ is a *discrete-time event log* such that
  - $\forall \pi \in \Pi$ and π is a discrete-time event trace.

As an example of an discrete-time event log take Table 3. The discrete-time event log contains 17 discrete-time events for 4 different traces which corresponds to four different orders. Order one and three was handled by Avery (avr), order two by Brian (bri) and order four by Charlie (cha). Notice that when a carrier is moved multiple M events are registered. Firstly, the M action type indicates that an entire carrier is moved; hence, the items on a carrier are removed or placed simultaneously. Secondly, in LOGIA both the removal of items from the old location and placement into the new location are logged when the items have been relocated. Consequently, moving a carrier results in a sequence of M events with the exact same delay.

Type	Location	Item	Quantity	Worker	Order number	Time
C	859-05-03-B	47	33	avr	1	0
O	859-05-03-B	47	-10	avr	1	11
C	444-01-03-B	15	4	bri	2	19
O	444-01-03-B	15	-4	bri	2	33
M	858-03-11-A	74	-32	avr	3	71
M	859-01-11-B	74	32	avr	3	71
M	858-03-11-A	97	-15	avr	3	71
M	859-01-11-B	97	15	avr	3	71
O	853-01-01-C	46	-109	bri	2	118
O	227-05-03-B	8	-23	avr	3	157
M	308-04-03-C	16	-87	cha	4	192
M	441-18-03-B	16	87	cha	4	192
M	308-04-03-C	69	-94	cha	4	192
M	441-18-03-B	69	94	cha	4	192
O	441-12-04-A	11	-98	bri	2	201
O	441-18-03-B	16	-45	cha	4	270
O	439-05-03-A	17	-31	cha	4	368

Table 3: *Example of a discrete-time event log based on our case study*

7. Swap Suggestions

In the following section we provide an overview of our swap suggestion algorithm and its most central element; the cost function. First we outline the goal of the swap suggestion algorithm followed by a description of several functions used to extract information from event logs. We then introduce discounting in order to

assign preference to recent events. Finally, we conclude by defining the cost function and its elements, and contextualise the cost function in our swap suggestion algorithm.

7.1. Goal

The goal which our swap suggestion algorithm must achieve has been previously stated in Section 5.2 and is in essence what Gu et al. [15] refers to as the storage location assignment problem. This problem is not concerned with general storage issues such as how many items should be kept in storage, but solely regards the assignment of storage locations to items. The authors emphasise two major criteria with regards to this problem. The first, called *storage efficiency*, corresponds to the holding capacity of a location, and the second, *access efficiency*, corresponds to the amount of resources spend on taking items from locations. We shall refer to these two criteria collectively as the *expenditure* and use it to indicate the expenditure associated with two given swap candidates post-swap.

Swapping two candidates can result in either an increase, decrease, or breakeven in expenditure. However, we must also consider the time it takes for a worker to actually perform the swap, i.e. the *swapcost*. Since the swapcost is always an expense this must be added to the expenditure resulting from a swap which results in a *cost*. The cost may be positive or negative, but it is always the desirable outcome to get a negative cost i.e. we can safely disregard all swaps which result in a positive cost.

7.2. Item Average

The storage efficiency of a given reserve location, in relation to the the execution of the action type sequence of RO followed by RI, i.e. the replenishment action, is equal to the number of times the worker can access the location before it is emptied. When computing the storage efficiency for a given item we base it on the average number of units taken when replenishing said item. In Definition 7.1 we present the equation which computes the average items taken, where we have a timed event log Π being a sequence of actions, as:

$$\begin{aligned} \Pi = \; & ((t_1, l_1, i_1, n_1, w_1, o_1), d_1) \\ & ((t_2, l_2, i_2, n_2, w_2, o_2), d_2) \\ & \vdots \\ & ((t_n, l_n, i_n, n_n, w_n, o_n), d_n) \end{aligned} \tag{1}$$

In the numerator of Definition 7.1 we calculate the sum all items i occurring in Π where the type is *RO*, and in the denominator we calculate the sum of *RO* containing some quantity of item i over the same conditions.

Definition 7.1 (Item Average)**.** Given a discrete-time event log Π of the form as show in Equation (1), we define the average number of items $\bar{i}$ accessed per replenishment as:

$$item\_avg(\bar{i}) = \frac{\sum\limits_{\substack{j=1 \\ t_j = RO \\ \bar{i} = i_j}}^{n} n_j}{\sum\limits_{\substack{j=1 \\ t_j = RO \\ \bar{i} = i_j}}^{n} 1}$$

7.3. Time Discounting

In Definition 7.1 each observed *RO* type in the discrete-time event log contributes equally. As a result hereof observations made one month ago are as indicative of the current replenishment trend as those made during the previous week. However, this is intuitively not the case since customer demand changes e.g. as seasons change or other events occur such as school start. Frederick et al. [23] introduces the notion of *time discounting* which encompasses any reason to care less about future consequences or any factor that diminish the expected utility they generate. In the context of our work we care less about what happened a month ago since it is less of a causal factor into the future than what happened a week ago.

To incorporate time discounting into our swap suggestion algorithm we extend Definition 7.1. Consequently, we have the extended version as seen in Definition 7.2, where we summarise over n time units. Furthermore, the extension puts a lower weight to the average, as we get further from the newest discrete time unit, where the discrete time unit for Av Form's logs is is seconds, and as the use of seconds would yield rather small values, we reduce the discrete time unit to represent days instead.

Definition 7.2 (Discounted Item Average)**.** Given an event log Π of the form as shown in Definition 6.6, we define the discounted item average for $\bar{i}$ as:

$$disc\_item\_avg(\bar{i}) = \sum_{j=1}^{n} \left(\frac{\sum\limits_{\substack{k=1 \\ t_j=RO \\ \bar{i}=i_j}}^{n} n_j}{\sum\limits_{\substack{k=1 \\ t_j=RO \\ \bar{i}=i_j}}^{n} 1} * \frac{1}{\left(\frac{d_n - d_j}{60*60*24}\right)} \right)$$

7.4. Cost

As mentioned we aim at decreasing the cost resulting from swapping two locations. This cost which we will elaborate on and define in the following is based on the expenditure of two swap candidates post-swap and the cost of swapping them which we refer to as swapcost.

7.4.1 Expenditure

In Definition 7.3 we define the equation to calculate the expenditure resulting from swapping two locations. The expression consist of two major terms, each encapsulated in parenthesis, separated by the minus operator. The first term calculates the storage cost when two locations have been swapped, whereas the second calculates the same storage cost but without the two locations having been swapped.

Computing the expenditure associated with a location is performed as follows: For each item i on a location l, multiply the time it takes to access l with the number of times i can be accessed, based on the average items taken when accessing it, before emptying the carrier. This, in essence, corresponds to multiplying the access efficiency of l with its storage efficiency where the storage efficiency is the number of times we can replenish item i from l.

Definition 7.3 (Expenditure)**.** Given a warehouse WH = (*L, I, Dist, Lift, Swapable*) in a configuration C, we define the effect of swapping locations l_1 and l_2 as:

$$expenditure(l_1, l_2) = \Big(\sum_{i \in l_2} Lift(l_1) * \frac{Config(l_2)(i)}{item\_avg(i)} + \sum_{i \in l_1} Lift(l_2) * \frac{Config(l_1)(i)}{item\_avg(i)} \Big) - \Big(\sum_{i \in l_2} Lift(l_2) * \frac{Config(l_2)(i)}{item\_avg(i)} + \sum_{i \in l_1} Lift(l_1) * \frac{Config(l_1)(i)}{item\_avg(i)} \Big)$$

To emphasise on the result, the expenditure will be negative if the new configuration, with the swapped locations, yields a lower value than the configuration prior the swap.

7.4.2 Swapcost

Let us first examine the workflow that is used in the computation of the swapcost. Starting at the known location l_1, the worker retrieves the carrier on location l_1 which he then brings to reserve location l_2. The worker then temporarily places the l_1 carrier on the floor while retrieving the carrier of l_2. Subsequently, the worker places the l_1 carrier into the now empty location l_2, and returns to the location l_1 placing the l_2 carrier into that location, and hence completing the swap. We formalise this workflow as the swapcost function as seen in Definition 7.4.

Definition 7.4 (Swapcost)**.** Given two locations l_1 and l_2 from a warehouse $WH = (L, I, Dist, Lift, Swapable)$ to be swapped, we define the swapcost as follows:

$$swapcost(l_1, l_2) = 2 * (Lift(l_1) + Dist(l_1, l_2) + Lift(l_2))$$

We consider the swapcost since the reduction to the expenditure may be so small, that the time invested in performing the swap could make the expenditure reduction insignificant.

7.4.3 Cost Function

With the formal definition of both the expenditure and swapcost we formally define the cost as seen in Equation (2). As mentioned previously the cost signifies the expenditure added with the swapcost. If the reduction in expenditure is greater than the swapcost we will get a negative cost i.e. the time gained from swapping two locations outweigh the cost of swapping them.

$$cost(l_1, l_2) = expenditure(l_1, l_2) + swapcost(l_1, l_2) \tag{2}$$

7.5. Swap Suggestions Algorithm

With the swap function defined in Equation (2) we can then compute the swap suggestions as seen in Algorithm 1. In the swap suggestions algorithm we have to compare every location to every other location, hence if we have n locations, we must perform a total of $\frac{n^2-n}{2}$ calculations for possible swap candidates, which

results in a complexity of $O(n^2)$.

Algorithm 1: Swap suggestions algorithm

```
1  function swap_suggestions(L, Π)
       input  : A set of locations L and a log Π
       output : A list of swap suggestions with their respective cost
2      let L = {l_1, l_2, ..., l_n}
3      let result be a list of location pairs and a cost value
4      for l_outer = l_1..l_n − 1 do
5          for l_inner = l_outer + 1..l_n do
6              candidate = cost(l_outer, l_inner)
7              if candidate <= 0 then
8                  add ((l_outer, l_inner), candidate) to result
9      sort result on candidate ascending
10     return result
```

8. Associating Time to Action Types

In Table 1 we have described different actions a worker can perform in a warehouse, and in Definition 6.1 we included the functions *Dist* and *Lift*. These actions and functions share a common resource, which is *time*. In this section we introduce a set of actions, and we introduce and apply a process for appending time estimates onto this set of actions.

8.1. A Worker's actions

A worker's daily tasks involve a number of rudimentary *worker actions*. We provide the following brief description of worker actions:

Aisle Traversing A worker drives on a forklift from one end of an aisle to the opposite end of that same aisle

Corner Turning A worker turns the corner of an aisle

Between Aisles Traversing A worker drives, after having exited an aisle, in a straight line from one aisle to the entrance of another aisle performing no corner turning under the trip

Lift Accessing A forklift operator either; lifts a carrier from the ground up to a reserve location also bringing the forks down again, or starts with empty forks at ground level and takes a carrier from a reserve location down to the ground.

8.2. Associating Time to Worker Actions

By observing the logs in LOGIA, we are not able to gather information of the time we workers spend on the different worker actions, and we have no other logged measure of how the workers at Av Form move around in the warehouse. Instead, during a visit to the warehouse, we gathered time measurements of a single worker operating a forklift. The measurements were gathered with minimal interference from the observers, which coupled with the discrepancy between the walking speed of the observers, and the driving speed of the forklift resulted in sparse measurements. However, as a starting point for a solution these measurements suffice as approximations.

Furthermore, the measurements for both aisle traversing and corner turning were carried out sporadically since the values measured did not fluctuate significantly—the observed fluctuation did not exceed more than

one second. The measured times for these two worker actions are as follows: Aisle traversing seven seconds and corner turning time two seconds. The remaining observations had more fluctuations, hence we will look at these observations in some more detail.

From	To	Time(sec)	Aisle Distance	Time per Aisle
312	308	9	2	5
312	427	19	4	5
423	434	20	5	4
304	424	8	7	1
312	427	19	4	5
312	308	9	2	5
304	428	16	9	2
420	424	8	2	4

Table 4: *Rounded measurements of: Between aisles traversing*

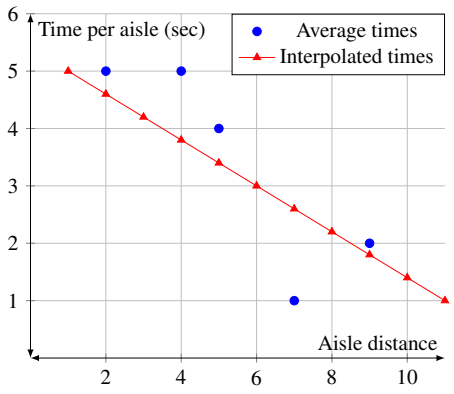


Figure 4: *Average and interpolated times for between aisles traversing*

From the between aisles traversing measurements seen in Table 4, we see the measured time per aisle mostly is five seconds for aisle distances 2 and 4. We consider the single aisle distance on 5, though seeming reasonable, as noise, because a line-of-sight issue was encountered during the measurements. The remaining aisle distances of 7 and 9 were measured with a low time per aisle average.

From the observations, we deduce that longer distances result in lower time per aisle averages, which is also intuitively expected as the forklift operator will achieves higher speeds over longer distances. The increase in time per aisle from distance 7 to 9 is explained from the layout of the warehouse, as the latter distance included passing a packing area with other workers in near proximity, requiring a lower speed.

For the purpose of constructing unkown data points we use the technique of interpolation where an approximating function is used to construct new data points such that they adhere perfectly to the, often, unknown function of the original data set [24]. In order to generalise the collected data for between aisles traversing we abstract away from the behaviour explaining x=7 and x=9. We also assign the slowest time per aisle of five seconds to x=1 and the fastest time per aisle of one second to x=11 since this is the longest aisle distance at Av Form. Applying linear interpolation using the two datapoints (1,5) and (11,1) results are shown in Figure 4, in which each aisle distance incrementation results in a reduction in time per aisle of four milliseconds.

Row	Time(sec)	Avg. time
1	24, 25, 24, 25, 31	25
2	30, 30, 36, 36, 32 27, 31	32
3	33, 27	30
4	42, 43, 44, 41	42

Table 5: *Rounded measurements of: lift accessing*

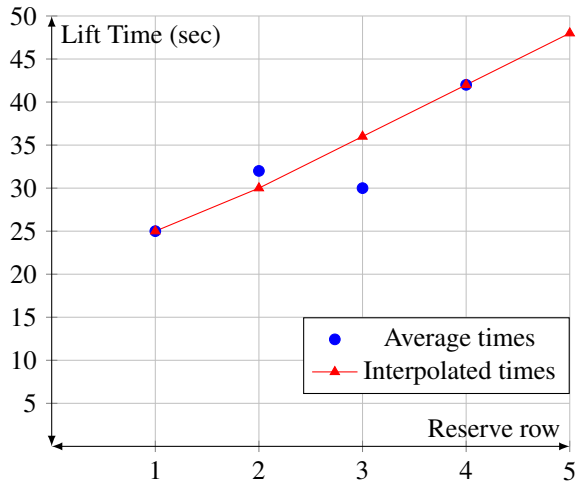


Figure 5: *Average and interpolated data for lift accessing, based on Table 5, where row one is the lowest row*

The time measurements gathered for the lift accessing can be seen in Table 5. Notice that the average lift time for row two and three decreases from 32 to 30. This is due to the measurment method used where we used the row number and not physical height of the row; hence a row number for different racks can vary in physical height. In order to remedy this erroneous data we decided to interpolate using only the lift times for row one and four, as these measurements were more consistent. From the interpolated times, shown in Figure 5, we observe that it takes five to six seconds for the worker to lift the forks from a row to its adjacent rows. To conclude, we present the time values for the different worker actions in Table 6, and we show how the time values can be used to create heuristic functions for Av Form to calculate the $Dist(l_1, l_2)$ and $Lift(l_1)$ functions presented in Definition 6.1 for a warehouse.

Aisle Traversing (AT)	7 seconds	
Corner Turning (CT)	2 seconds	
Between Aisles Traversing (BAT)	aisle distance	time (sec)
	1	5
	2	9
	3	13
	4	15
	5	17
	6	18
	7	18
	8	18
	9	16
	10	14
	11	11
Lift Accessing (LA)	row height	time (sec)
	1	25
	2	30
	3	36
	4	42
	5	48

Table 6: *Fixed time metric for each of the measured actions with rounded time values*

The $Lift(l_1)$ can be calculated as seen in Equation (3), where *row_height* is specified for the given location, as the number of shelves counted from the lowest reserve location which is 1, the shelf above is 2, and so forth. The pick locations are not considered as these can be reached from ground level without the use of a forklift. In Equation (3) the constant value of 19 seconds is a baseline for the duration of the lift time, where each increase in row height adds another 6 seconds.

$$Lift(l_1) = 19 + row\_height * 6 \tag{3}$$

The Equation (4) is used to calculate the $Dist(l_1, l_2)$, where x is the amount of traversed aisles, and y is the amount of involved corner turnings. Furthermore, the heuristics of this equation include BAT, as we assume a straight line when traversing the warehouse, where only one BAT is included for the aisle distance between the two involved locations.

$$Dist(l_1, l_2) = x * AT + y * CT + BAT(l_1, l_2) \tag{4}$$

9. Implementation in LOGIA

As previously mentioned, two stakeholders are involved in this project: The business Av Form which has a warehouse, and Logimatic whom provides Av Form with a warehouse management system (WMS) called LOGIA. We have implemented the cost function in Equation (2) on page 14 into LOGIA, and in this section we will present the most prominent aspects of the user interface which we have implemented, followed by the underlying implementation of the cost function.

LOGIA is a WMS used in multiple warehouses in different countries. Logimatic have some predefined guidelines on how functionlity and the userface is designed and implemented, but we still have some influence on how we wish to design the implementation. LOGIA is a *Windows Presentation Foundation* (WPF)

application [25], which follows a *Model View View-model* (MVVM) pattern [26]. We will not go into details on the underlying implementation of the user interface, but rather describe it from screenshots in this section.

Through our theory, we support two scenarios for for suggesting locations to swap, either where we know one location we would like to swap, or where we consider all possible location pairs. The benefit from the former scenario is that it can be implemented in the workers daily work where, when the worker is accessing one location, we can suggest possible other locations to swap said location with. The latter scenario is more applicable in situations where the worker would be otherwise idle, and we can suggest a set of location pairs ideal for swapping.

We emphasised our desire for a solution that included both these possibilities, but trough a discussion with Av Form, and from our visit at the warehouse, we see that knowing one location beforehand, though possible, would not result in an effective workflow, as it did not seem feasible having to travel with the additional carrier while performing the ordinary replenish task. Hence our implementation only support knowing neither of the locations beforehand.

9.1. User Interface

In Figures 6 through 8 we show screenshots of what Logimatic refers to as screens. We present the screens in the same order as a worker is presented with the screens, when she has to perform a swap task.

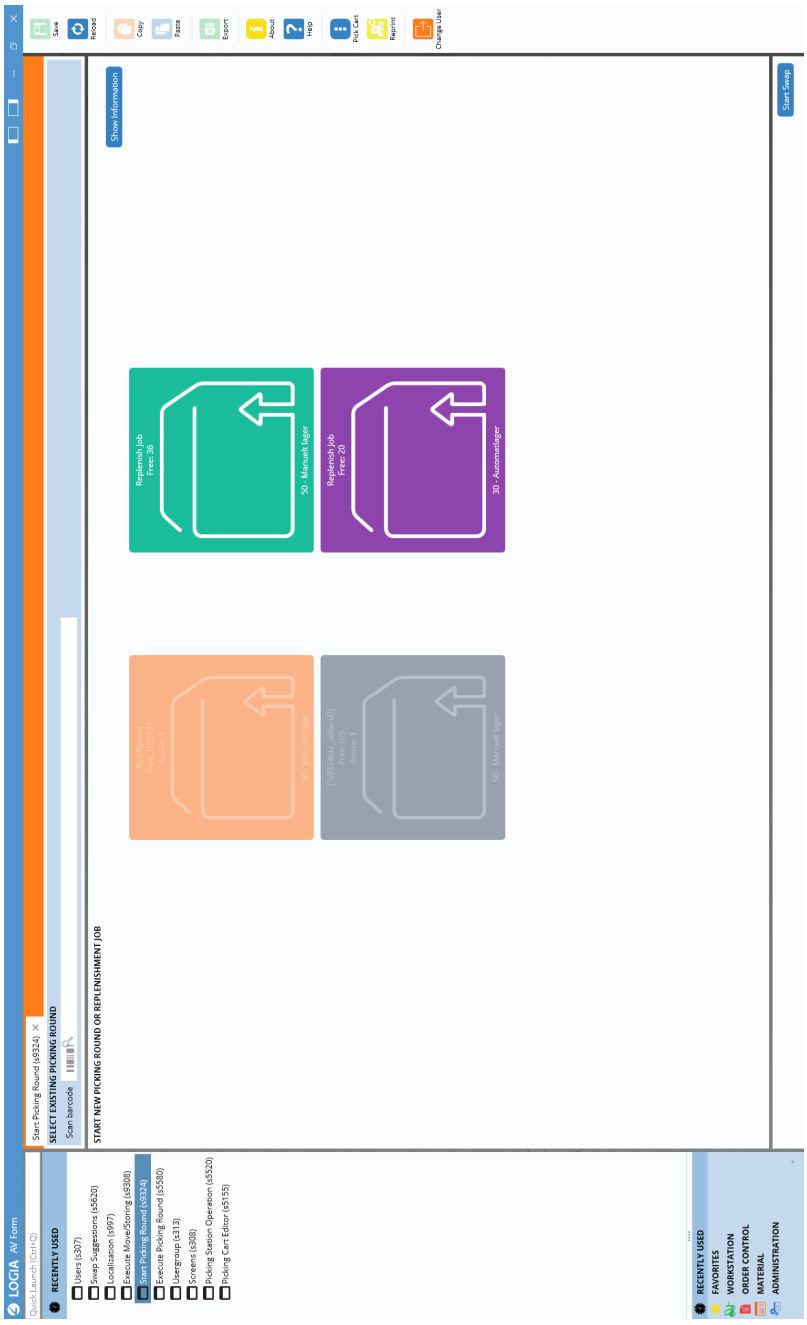


Figure 6: *Screenshot of LOGIA screen: Start Picking Round*

9.1.1 Screen: Start Picking Round

The workers performing existing replenish order tasks prior to our implementation accessed the screen *Start Picking Round* shown in Figure 6. We have chosen to extend the screen with the button *Start Swap* placed in the lower right side of the screen, which is used to open the screen *Swap Suggestions*. This button is seen as an extension, as it would already be possible to access the screen from the menu option on the left side of the screen. But we have chosen to include the extra button as a reminder of the new option.

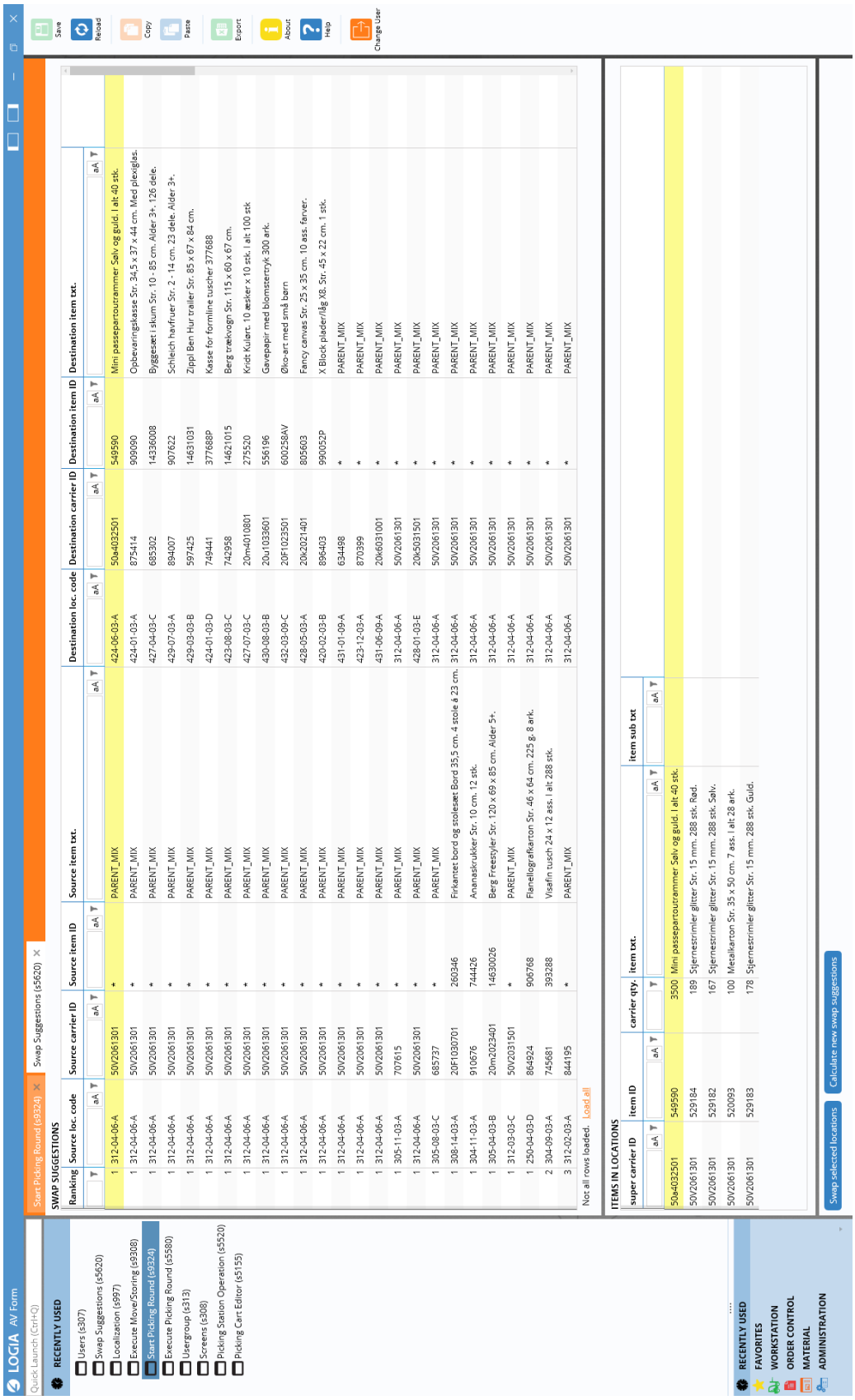


Figure 7: *Screenshot of LOGIA screen: Swap Suggestions*

9.1.2 Screen: Swap Suggestions

The screen shown in Figure 7 Is our major extension to the user interface, as the screen did not exist before hand. The screen presents the calculated swap suggestions using two tables which are implemented with a master-detail relation.

Master-Detail Tables

The master table named "SWAP SUGGESTIONS" shows the list of location pairs we suggest to swap, prioritised on how much the swap will reduce the calculated expenditure, with the greatest reduction shown in the top. We include the first column titled "Ranking" for two reasons: Multiple location pairs may have the same cost result; hence, we add the same ranking for these pairs and a worker may choose to change the sorting in the table, here the ranking helps to guide the worker to not pick a swap suggestion which has a high ranking.

We as software developers lack the domain knowledge that the warehouse workers have; hence, we include the detail table named "ITEMS IN LOCATIONS". This table updates when a worker selects a row in the master table showing certain details of the items on the two selected swap locations. The worker may then review the item details and decide whether or not to commence with the selected swap suggestion.

The Buttons

We provide two buttons, at the bottom left of this screen, the first titled "Calculate new swap suggestions" and the second called "Swap selected locations". The former button was implemented because the process of calculating the cost of all the locations is cumbersome and may take several seconds to complete. Hence we do not want to force the workers to wait this long duration every time they have completed a swap. But we invite them to perform the calculations as frequently as possible.

The latter button, as the title implies, initiates the process of swapping the two selected locations. Using existing functionality in LOGIA, we open the next screen; *Execute Move/Storing* shown in Figure 8 while deactivating all functionality in the *Swap Suggestions* screen. While initiating the new screen, we create a sequence of move transactions, and we place the involved locations in an ignore list to ensure that other workers does not initiate work on the same locations.

The *Execute Move/Storing* screen may be closed due to different reasons. Upon closing the screen different things happens in the *Swap Suggestions* screen. We will explain these events before explaining the *Execute Move/Storing* screen.

Execute Move/Storing may be closed due to two different outcomes; either the swap is completed, or it is cancelled. If the swap is cancelled, the screen is updated to accommodate for possible changes performed by other workers in the warehouse, and the created move transactions are deleted. If the swap is completed, we also refresh the screen, but the locations are kept in the ignore list. This is done due to the decision not to automatically recalculate the swap suggestions after each swap. Where the ignored locations are again included after a recalculation has been performed.

Figure 8: *Screenshot of LOGIA screen: Execute Move/Storing*

9.1.3 Screen: Execute Move/Storing

The screen *Execute Move/Storing* shown in Figure 8 is opened when "Swap selected locations" is clicked, as previously explained. All elements shown in the screenshot are part of the existing LOGIA functionality, but we will include some more changes later in the description. First we explain the expected sequence of actions, followed by more details about some of the actions.

Action sequence

We discussed some implementation strategies with Logimatic, to either introduce a new action where all needed actions are included, or use the existing actions to achieve the swap. The drawback of the first implementation is an increase in the implementation complexity, and more importantly that all actions associated with the swap event would be logged at the same time, such that details on the duration of completing the intermediate steps are lost. The second implementation would be more simple to implement, as we can use the existing MoveC(l_1, l_2) function, by introducing a temporary location l_t such that we can perform the sequence MoveC(l_1, l_t) $\rightarrow$ MoveC(l_2, l_1) $\rightarrow$ MoveC(l_t, l_2). The drawback of the latter implementation is that the worker will have to execute more actions, such as scanning barcodes of each location as they are accessed, but this is already a requirement for most actions in LOGIA, hence we do not see it as an significant drawback and went with the latter solution of using existing actions, which will carry additional information, to enable us to identify the MoveC(l_1, l_2) functions used as part of a sequence to perform a swap.

Introducing the temporary location

When the swap is initiated, we create two move actions, shown in Figure 9. This allows the worker to pick which location to begin at, e.g. based on the distance to either location. Given the two locations '312-04-06-A', and '424-06-03-A', henceforth referred to as l_1, l_2 respectively. The two generated actions are MoveC(l_1, l_2) and MoveC(l_2, l_1). If the worker wishes to execute l_1 as the first move, she first has to move l_2 to a temporary location. When performing this move, LOGIA registers that it is not the original location which is scanned and the dialogue in Figure 10 is shown. This dialogue is the last change we have made to the user interface, as there originally was no option for "Temporary location". If the worker selects this option, a new moveC action is created, with the scanned location as the new source location, and the original destination location, still being the designated destination location, and the old MoveC action is now replaced with the new action.

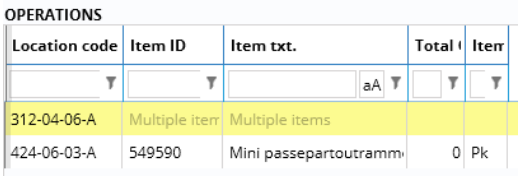


Figure 9: *snippet with two move actions*

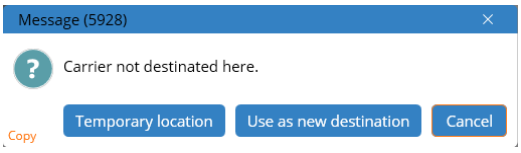


Figure 10: *Dialogue shown in s9308*

9.2. Database Implementation

The underlying database used in LOGIA is an Oracle database [27], which is utilised using PL/SQL; "*a procedual language designed specifically to embrace SQL statements within its syntax*" — as stated in [28]. We will not go into details with the two environments, but have to include a well discussed bottleneck of context switching [29]. SQL and PL/SQL are two different environments, and it is sometimes necessary to switch between the two environments. Performing this context switching only has a rather small overhead, but if the context switching is done often the overhead will become noticeable. We first became aware of this bottleneck during the implementation, and had to make some changes in order to achieve an increase in performance.

One approach we have made to reduce the context switching, is to divide the calculation of the swap expenditure into two; preprocessing, and performing the pairwise comparison of locations.

9.2.1 Preprocessing

In PL/SQL we can use cursors for executing queries where we expect to retrieve multiple rows [30]. These queries are SQL SELECT statements, and are primarily used to fetch data into an array in PL/SQL. This introduces context switching, hence we wish to reduce the number of times we fetch data from cursors into arrays.

We primarily use two cursors, one to fetch data from one of four logs in LOGIA, namely the tracelog, and one to fetch information from the locations. We will refer to these cursors as *c_translog* and *c_location* respectively.

c_translog cursor

We use the c_translog cursor for gathering information from the translog, and we query two attributes; *item_no* and *item_avg*. We use entities other than the translog to get the item_no as we are interested in all items, not restricted to those present in the translog. Listing 1 shows a snippet of the selected attributes in the c_translog cursor, where for each day, and each item:

- *sum_item_qty* is the total quantity of the item retrieved for replenish,
- *cnt_order* is the total amount of replenish orders placed for the item,
- *trunc(SYSDATE)* which gets the current date, without time information, and
- *date_translog* is the date, for which we retrieve the *sum_item_qty*, and *cnt_order*

We use this to calculate the *disc_item_avg* (Listing 1) for all items, where we set the value to 0 if no occurrences of the item were found in the translog.

```
1 CURSOR c_translog IS
2 SELECT
3   carrier.item_no,
4   NVL(ABS(SUM(sum_item_qty/cnt_order / (trunc(SYSDATE) - date_translog + 1))),0) AS item_avg
5 FROM carrier
6  . . .
```

Listing 1: *disc_item_avg(i) implementation*

We fetch the cursor c_translog into an array where we index each item_avg on the item_no, which later enables us to perform a single lookup and remove the need of a loop.

c_location cursor

The cursor c_location is more cumbersome than the other, as it is used to fetch more information, including the current quantity of items on each location, which items are on which locations, and size information on both carriers and locations.

We use the size information to determine which carriers fit into which locations, to avoid suggesting swaps that cannot physically be performed.

Next we populate an array *a_location* which holds information on each location, and a sub array where we include the item information of all items in the location, from the array populated by the c_translog cursor.

9.2.2 Pairwise comparison of locations

Early in the implementation, we had some processes which heavily increased the calculation time. The major increase to duration happened because we performed the *disc_item_avg* within the loop, which introduced context switching for every location pair. In the final implementation, we have two nested for loops, looping over the same array a_location, on two different indices. Hence, we can use the two indexes to get information on two locations in the array, and we can get all needed information for the two locations using direct lookups, reducing both the need for further looping, and context switching.

9.2.3 Handling Concurrent Work

It is possible for two workers to execute swaps at the same time, as we briefly covered in Section 9.1. We also had to consider this in the implementation into PL/SQL, which we cover in the following.

Ignore table

We maintain an *ignore_table* for two reasons. Firstly, when we calculate the swap candidates, it is likely that a location is in multiple rows of the result set as there may be multiple candidates of locations to swap with. Since we do not perform a recalculation of the swap candidates every time two locations are swapped, it is important that we do not show any rows with the involved locations. We do this by adding the involved locations to the ignore_table when a worker initiates the swap. The locations are kept in the ignore_table until either; the swap is cancelled before completion, or the swap expenditure is recalculated.

The second reason for the ignore_table is also related to the recalculation of the swap candidates. If we have two workers, *w1* and *w2*, performing swaps and *w1* is currently swapping two locations, while *w2* is initiating a recalculation. It is important that the two locations being swapped by *w1* is kept in the ignore list, as the recalculation includes the locations prior to the swap. We do this by adding a boolean attribute *is_reserved* to locations in the ignore_table. When the swap is initiated we set the boolean to true, until the swap is completed. When we perform the recalculation, we can then simply delete all rows from ignore_table where is_reserved is false.

Process locking

Two workers may not initiate the recalculation of swap candidates at the same time, as the procedure modifies tables, and trying to call the procedure when it is already running, would result in an error. To solve this issue we use existing functionality in LOGIA, where we try to fetch a lock for the procedure. If the procedure is not locked beforehand, we lock the procedure, executes it, and release the lock afterwards. If the procedure is locked when we try to fetch the lock, we throw an error in LOGIA saying the process is running elsewhere.

Part II
Process Discovery

10. Preliminaries

In this section we present the theory of extended timed-arc Petri nets as well as a subclass hereof called extended timed-arc workflow nets. We present this theory as preliminary knowledge, as it was part of our previous work in [11], and since we do not develop the theory as part of our contribution but use it as the foundation of our contribution.

10.1. Extended Timed-Arc Petri Nets

In the following we introduce the definitions for *timed transition systems* (TTSs) and *extended timed-arc Petri nets* (ETAPNs), as given by Mateo et al. [31, pp. 93–95], and we present our extension of their ETAPN definition, which includes a labelling function L. Furthermore, we also include several concepts from Mateo et al. [31, pp. 93–95] which accompanies the two definitions such as the preset and postset of the transitions and places of an ETAPN.

Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, $\mathbb{N}_0^\infty = \mathbb{N}_0 \cup \{\infty\}$, and let $\mathbb{R}^{\geq 0}$ denote the set of all nonnegative real numbers. Then a TTS can be defined as follows:

Definition 10.1 (Timed Transition System)**.** A TTS is a triple $(S, Act, \rightarrow)$, where

- S is the set of states,
- Act is the set of actions,
- and $\rightarrow \subseteq S \times (Act \cup \mathbb{R}^{\geq 0}) \times S$ is a transition relation which we will write as $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$.

Mateo et al. [31] briefly introduces some additional basic concepts related to TTS's.

- A transition can be either
  - a *switch transition* if $a \in Act$, or
  - a *delay transition* if $a \in \mathbb{R}^{\geq 0}$.
- The set of *well-formed closed time intervals* as $\mathcal{I} \stackrel{\text{def}}{=} \{[a, b] \mid a \in \mathbb{N}_0, b \in \mathbb{N}_0^\infty, a \leq b\}$.
  - and its subset, which is applied in age invariants, $\mathcal{I}^{inv} \stackrel{\text{def}}{=} \{[0, b] \mid b \in \mathbb{N}_0^\infty\}$.

From now on we will, in the presented work, assume a fixed set Act.

Definition 10.2 (Extended Timed-Arc Petri Net)**.** An *extended timed-arc Petri net* (ETAPN) is a 10-tuple $N = (P, T, T_{urg}, IA, OA, g, w, Type, I, L)$ where

- P is a finite set of *places*,
- T is a finite set of *transitions* such that $P \cap T = \emptyset$,
- $T_{urg} \subseteq T$ is the set of *urgent transitions*,
- $IA \subseteq P \times T$ is a finite set of *input arcs*,
- $OA \subseteq T \times P$ is a finite set of *output arcs*,
- $g : IA \rightarrow \mathcal{I}$ is a *time constraint function* assigning guards to input places such that if $(p, t) \in IA$ and $t \in T_{urg}$ then $g((p, t)) = [0, \infty]$,
- $w : IA \cup OA \rightarrow \mathbb{N}$ is a function assigning *weights* to input and output arcs,
- $Type : IA \cup OA \rightarrow$ **Types** is a *type function* assigning a type to all arcs where **Types** $= \{Normal, Inhib\} \cup \{Transport_j | j \in \mathbb{N}\}$ such that
  - if $Type(z) = Inhib$ then $z \in IA$ and $g(z) = [0, \infty]$,
  - if $Type((p, t)) = Transport_j$ for some $(t, p) \in IA$ then there is exactly one $(t, p') \in OA$ such that $Type((p, t')) = Transport_j$,
  - if $Type((t, p')) = Transport_j$ for some $(t, p') \in OA$ then there is exactly one $(p, t) \in IA$ such that $Type((p, t)) = Transport_j$,
  - if $Type((p, t)) = Transport_j = Type((t, p'))$ then $w((p, t)) = w((t, p'))$,
- $I : P \rightarrow \mathcal{I}^{inv}$ is a function assigning *age invariants* to places, and
- $L : T \rightarrow Act \cup \{\tau\}$ is a *labelling function* assigning actions, including tau, to transitions.

As an example of an ETAPN take Figure 11 which consists of the $places = \{P1, P2, P3, P4, P5, P6, P7\}$ and $transitions = \{T1, T2, T3, T4\}$. Furthermore, all transitions have a label assigned to them e.g. transition $T1$ is assigned the label a. We will color τ-transitions, such as T4, black to clearly distinguish them from non-τ-transitions. The arcs $(P1, T1)$ and $(T1, P2)$ are both transport arcs indicated by the diamond shaped heads and form a pair indicated by : 1. Additionally, $(P1, T1)$ has a lower- and upperguard with the values 2 and 3 respectively. The input arc $(P4, T2)$ is an inhibitor arc indicated by the white bullet head, and the outgoing arc of $T2$ is assigned a weight of 4. Finally, the white dot in the transition $T4$ signifies an urgent transition, and $P7$ has an age invariant.

We will now further describe some notation in order to subsequently give the formal semantics of ETAPN. Let N be the ETAPN shown in Figure 11. By ${}^\bullet x \stackrel{\text{def}}{=} \{y \in P \cup T \mid (y, x) \in IA \cup OA,\ Type((y, x)) \neq Inhib\}$ we denote the preset of a transition or a place x, e.g. ${}^\bullet T2 = \{P2\}$. Likewise, we define the postset as

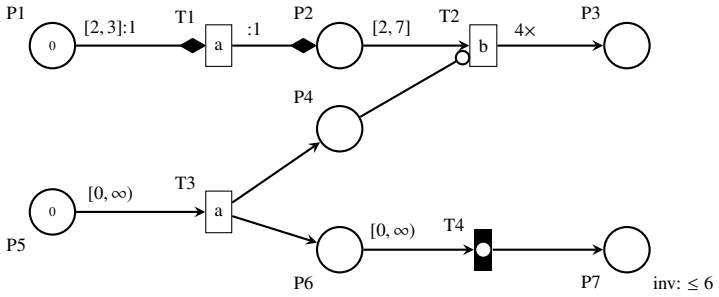


Figure 11: *Extended timed-arc Petri net example where τ-transitions are black*

$x^\bullet \stackrel{\text{def}}{=} \{y \in P \cup T \mid (x, y) \in (IA \cup OA)\}$ e.g. $T2^\bullet = \{P3\}$. We denote the set of all finite multisets over $\mathbb{R}^{\geq 0}$ as $\mathcal{B}(\mathbb{R}^{\geq 0})$. Then a *marking* M on N will be defined as a function $M : P \to \mathcal{B}(\mathbb{R}^{\geq 0})$ for which, for every place $p \in P$ and every token $x \in M(p)$ it holds that $x \in I(p)$ i.e. all tokens must adhere to their respective age invariants. By $\mathcal{M}(N)$ we denote the set of all markings for the net N.

The notation (p, x) denotes a token in place p with an age of $x \in \mathbb{R}^{\geq 0}$. It then follows that the multiset representing a marking M is noted as $M = \{(p1, x1), (p2, x2), \ldots, (p_n, x_n)\}$ where we have n tokens of ages x_i in places p_i. Finally, we denote the number of tokens in a place p as $|M(p)|$; consequently we define the size of a marking M as $|M| = \sum_{p \in P} |M(p)|$.

We also wish to introduce Mateo et al. [31]'s definition of *Enabledness*, as this covers when transitions are *enabled* and therefore can be *fired*. We include our extension to ETAPN, but this has no impact on the remainder of the definition.

Definition 10.3 (Enabledness)**.** Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I, L)$ be an ETAPN. We say that transition $t \in T$ is enabled in a marking M by the multisets of tokens $In = \{(p, x_p^1), (p, x_p^2), \ldots, (p, x_p^{w((p,t))}) \mid p \in {}^\bullet t\} \subseteq M$ and $Out = \{(p', x_{p'}^1), (p', x_{p'}^2), \ldots, (p', x_{p'}^{w((t,p'))}) \mid p' \in t^\bullet\}$ if

- for all input arcs except the inhibitor arcs, the tokens from In satisfy the age guards of the arcs, i.e.

$$\forall p \in {}^\bullet t.\ x_p^i \in g((p, t)) \text{ for } 1 \leq i \leq w((p, t))$$

- for any inhibitor arc pointing from a place p to the transition t, the number of tokens in p is smaller than the weight of the arc, i.e.

$$\forall (p, t) \in IA.\ Type((p, t)) = Inhib \Rightarrow |M(p)| < w((p, t))$$

- for all input arcs and outpout arcs which constitute a transport arc, the age of the input token must be equal to the age of the output token and satisfy the invariant of the output place, i.e.

$$\forall (p, t) \in IA.\ \forall (t, p') \in OA.\ Type((p, t)) = Type((t, p')) = Transport_j$$

$$\Rightarrow (x_p^i = X_{p'}^i \wedge x_{p'}^i \in I(p'))\text{for}1 \leq i \leq w((p, t))$$

- for all normal output arcs, the age of the output token is 0, i.e.

$$\forall (t, p') \in OA.\ Type((t, p')) = Normal \Rightarrow x_{p'}^i = 0 \text{ for } 1 \leq i \leq w((t, p')).$$

We now have that a given ETAPN N defines a TTS $T(N) \stackrel{\text{def}}{=} (\mathcal{M}(N), T, \to)$ for which states are markings, and switch and delay transitions are as follows.

- if $t \in T$ is enabled in marking M by the multisets of tokens In and Out then t can fire and produce the marking $M' = (M \setminus In) \uplus Out$ where $\uplus$ is the multiset sum operator and $\setminus$ is the multiset difference operator; we write $M \xrightarrow{L(t)} M'$ for this switch transition.
- A time *delay* $d \in \mathbb{R}^{\geq 0}$ is allowed in M if
  - $(x + d) \in I(p)$ for all $p \in P$ and all $x \in M(p)$, and
  - if $M \xrightarrow{L(t)} M'$ for some $t \in T_{urg}$ then $d = 0$.

  By delaying d time units in M we reach marking M' defined as $M'(p) = \{x + d \mid x \in M(p)\}$ for all $p \in P$; we write $M \xrightarrow{d} M'$ for this delay transition.

Mateo et al. [31] continues to define delay transitions in continuous time semantics i.e. $d \in \mathbb{R}^{\geq 0}$. However, since events recorded in an event log happen in discrete steps with respect to the time precision, we define delays in the discrete time semantics i.e. $d \in \mathbb{N}_0$.

We shall now introduce some final concepts in relation to markings. We let $\rightarrow \stackrel{\text{def}}{=} \bigcup_{t \in T} \xrightarrow{t} \cup \bigcup_{d \in \mathbb{N}_0} \xrightarrow{d}$. In a marking M we denote the set of all reachable markings as $[M\rangle \stackrel{\text{def}}{=} \{M' \mid M \rightarrow {}^*M'\}$. Moreover, by $M \xrightarrow{d,t} M'$ we denote that there is a marking M'' such that $M \xrightarrow{d} M'' \xrightarrow{t} M'$. Additionaly, we say that if in a marking M there are no $d \in \mathbb{N}_0$, $t \in T$ and another marking M' such that $M \xrightarrow{d,t} M'$ then M is a *deadlock*. A marking M may also allow for unbounded delays, more formally if for every $d \in \mathbb{N}_0$ we have $M \xrightarrow{d} M'$ for some M', in which case we call it *divergent*.

After having given the semantics for ETAPNs we can explore the behavior of the ETAPN in Figure 11 by playing the token game. Let us from the marking $M_0 = \{(P5, 0), (P1, 0)\}$ fire the following sequence of transitions: $T1, T3, T4$. First, we need to delay in order to enable and fire $T1$; $M_0 \xrightarrow{3,T1} M_1$. We are now in the marking $M1 = \{(P2, 3), (P5, 3)\}$ in which the age of the produced token in $P2$ is kept, as $T1$ is a transport arc. Immediately afterwards we fire $T3$ as its guard is satisfied; $M_1 \xrightarrow{T3} M_2$. In the new marking $M_2 = \{(P2, 3), (P4, 0), (P6, 0)\}$ $T2$ is disabled due to the inhibitor arc from $P4$ to $T2$ even though its guard is satisfied. The final transition $T4$ in our desired firing sequence is enabled and since it is urgent we cannot delay so we fire $T4$ immediately, $M_2 \xrightarrow{T4} M_3$. In the resulting marking $M_3 = \{(P2, 3), (P4, 0), (P7, 0)\}$ time can be delayed up to six time units due to the invariant in $P7$. Delaying more than four time units will result in the token in $P2$ not satisfying the guard on edge $P2$ to $T2$. However, given the marking $M3$ dalying for any time units does not impact the enabledness of $T2$ since the token in $p4$ will continue to inhibit it.

10.2. Extended Timed-Arc Workflow Nets

In [31, pp. 97–98], the authors also define timed-arc workflow nets and soundness of timed-arc workflow nets. Much like with the timed-arc Petri nets, we will also introduce their definitions here, with the small change that we use the ETAPN we extended to include the labelling function L.

A timed-arc workflow net (TAWFN) has exactly one input place and one output place, and is initialised by placing a token in the input place. An essential characteristic of any TAWFN, formally defined by the notion of soundness, is the guarantee that any execution trace can be extended such that it terminates with exactly one token in the output place.

Definition 10.4 (Extended timed-arc workflow net)**.** An $ETAPN\ N = (P, T, T_{urg}, IA, OA, g, w, Type, I, L)$ is called and *Extended Timed-Arc WorkFlow net* (ETAWFN) if

- there exists a unique place $in \in P$ such that ${}^\bullet in = \emptyset$ and $in^\bullet \neq \emptyset$
- there exists a unique place $out \in P$ such that $out^\bullet = \emptyset$ and ${}^\bullet \neq \emptyset$

- for all $p \in P \setminus \{in, out\}$ we have ${}^\bullet p \neq \emptyset$ and $p^\bullet \neq \emptyset$, and
- for all $t \in T$ we have ${}^\bullet t \neq \emptyset$.

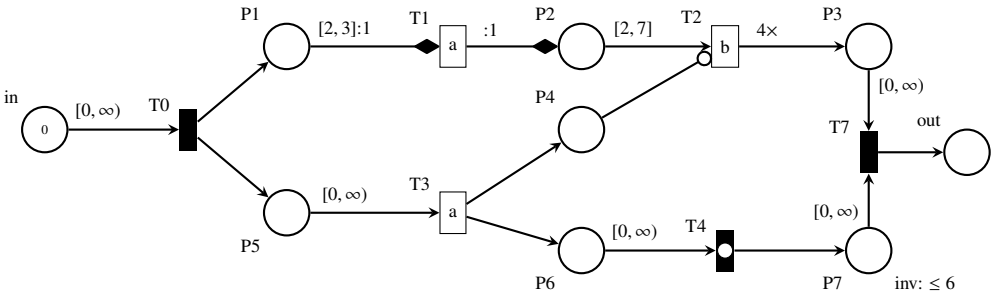


Figure 12: *Figure 11 transformed into a TAWFN*

As an example of an ETAWFN take Figure 12 which is in the marking $M_{in} = \{(in, 0)\}$ called the *initial* marking. A marking M is *final* if $|M(out)| = 1$ and for all $p \in P \setminus out$ we have $|M(p)| = 0$, i.e. M_{final} contains exactly one token in the place *out* and all other places are empty. There may be several final markings with different ages of the token in the place *out*.

Next we provide the formal definition of soundness as provided by Mateo et al. [31] where the authors have based said definition on the standard requirement for proper completion with regards to workflow nets [32, 33].

Definition 10.5 (Soundness of timed-arc workflow nets)**.** An extended timed-arc workflow net $N = (P, T, T_{urg}, IA, OA, g, w, Type, I, L)$ is sound if for any marking $M \in [M_{in}\rangle$ reachable from the initial marking M_{in}:

- there exists some final marking M_{out} such that $M_{out} \in [M\rangle$, and
- if $|M(out)| \geq 1$ then M is a final marking.

We now say that an ETAWFN is sound if after it is initiated it has the *option to complete* and subesquently it has *proper completion*. The option to complete requires that after a token of age zero has been placed in *in* it can be moved to the place *out*. Once the net has completed (a token has been placed in *out*) it must do so properly meaning it is guaranteed that all places except *out* is free of tokens.

The TAWFN in Figure 12 is currently not sound. Take the marking $M = \{(P3, 0), (P5, 6)\}$ in which the only enabled transition is $T3$. Firing $T3$ produces a token in the places $P4$ and $P6$ of age zero. Subsequently we can then fire $T4$ followed by $T7$ in order to place a token in *out*. However, this final marking does not satisfy the proper completion criterion as there still is a token in $P4$. To remedy this and make the TAWFN sound we may add an input arc from $P4$ to $T4$.

11. Discovery: Time Discrimination

In the following section, we look into the problem of time discrimination, related to process mining, and we look at two possible solutions, one where events are concatenated with previous events, and one where we divide time information info clusters.

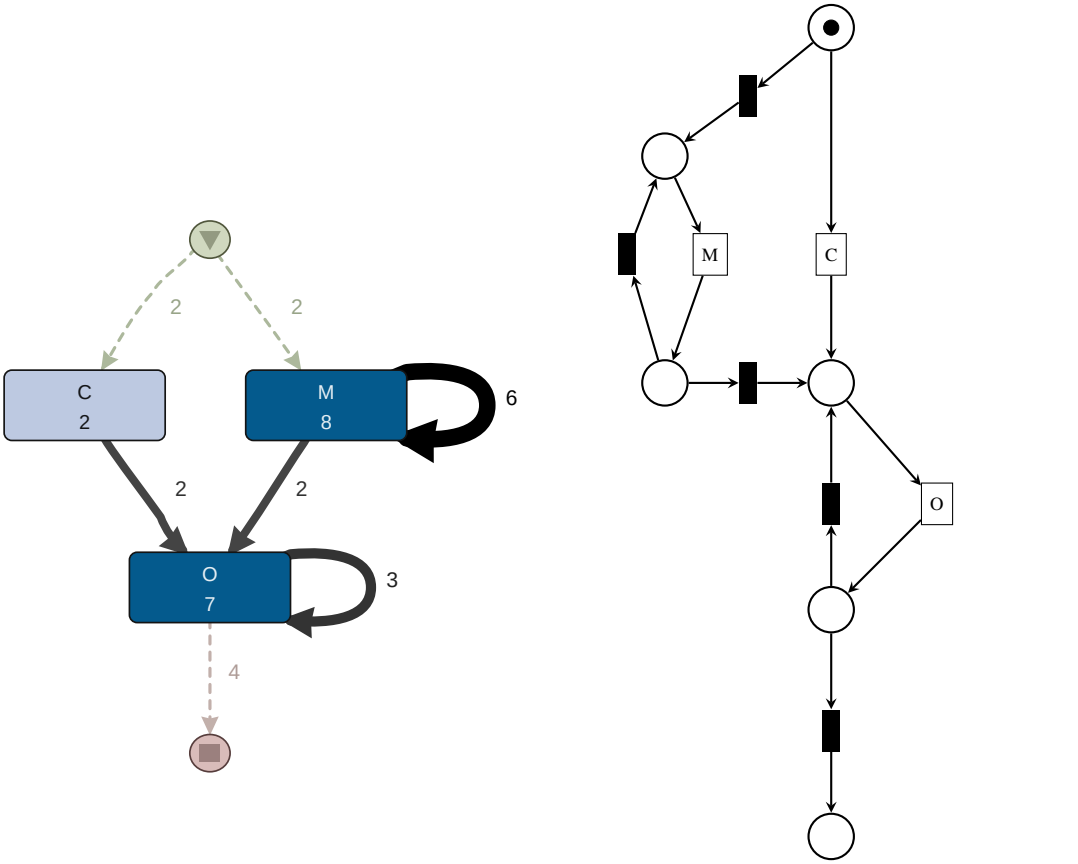
11.1. Time Dilemma

In our previous work [11] we presented a semi-automatic method for discovering an Extended Timed-Arc Workflow Net (ETAWFN) from a Discrete-Time Event Log (DTEL). Our method entailed manually combining the output of two process mining tools, ProM [18] for mining Petri nets and Disco [34] for mining process maps, in another tool called TAPAAL [35] by hand producing an ETAWFN. The issue of our method arises after we have constructed the ETAWFN, from the Petri net, and have to annotate it with time information from the process map. It is not a problem to construct the ETAWFN from the Petri net since the ETAWFN is a timed extension of the Petri net modeling language, but the process map is dissimilar to the Petri net modeling language. To give a brief example of this issue and the time dilemma it causes we use the DTEL previously shown in Table 3 on page 11.

By mining this DTEL in Disco and ProM we discover the process map shown in Figure 13a, and the Petri net in Figure 13b. Both models allow the same behaviour, but as mentioned previously differ in their notation as is evident from comparing the two. As the O action type most clearly illustrates the issue of our method we will focus on this one action type.

By inspecting the arcs going into the O node in the process map we can see the time metrics for; a C followed by an O, a M followed by an O, and an O followed by another O. If we look at the Petri net we see that the three aforementioned O sequences are encapsulated in the Petri net's behaviour. However, there are only one arc going into the O transition in the Petri net where we in the process map have three separate arcs for the O node. Consequently, when we annotate the Petri net with time information from the process map to produce an ETAWFN, we will have one guard for the O transition to express the three separate O arcs in the process map.

This in and of itself does not present a problem as long as there is a low spread in the observed delays for O. This, however, is not the case in our example DTEL in Table 3 where we have observed the following delays for O: $(11, 14, 78, 83, 85, 86, 98)$. In fact we have observed fast and slow O sequences: An O preceded by a C occurs with delay $(11, 14)$, an O preceded by a M with delay $(78, 86)$, and an O preceded by another O with delay $(83, 85, 98)$. It is from this pattern that the time dilemma arises; how do we express the O behaviour, with regards to time, recorded in the log in one guard? To unequivocally illustrate the issue of combining the observed O delays into one guard we use Figure 14a.



(a) *Process map from Disco*

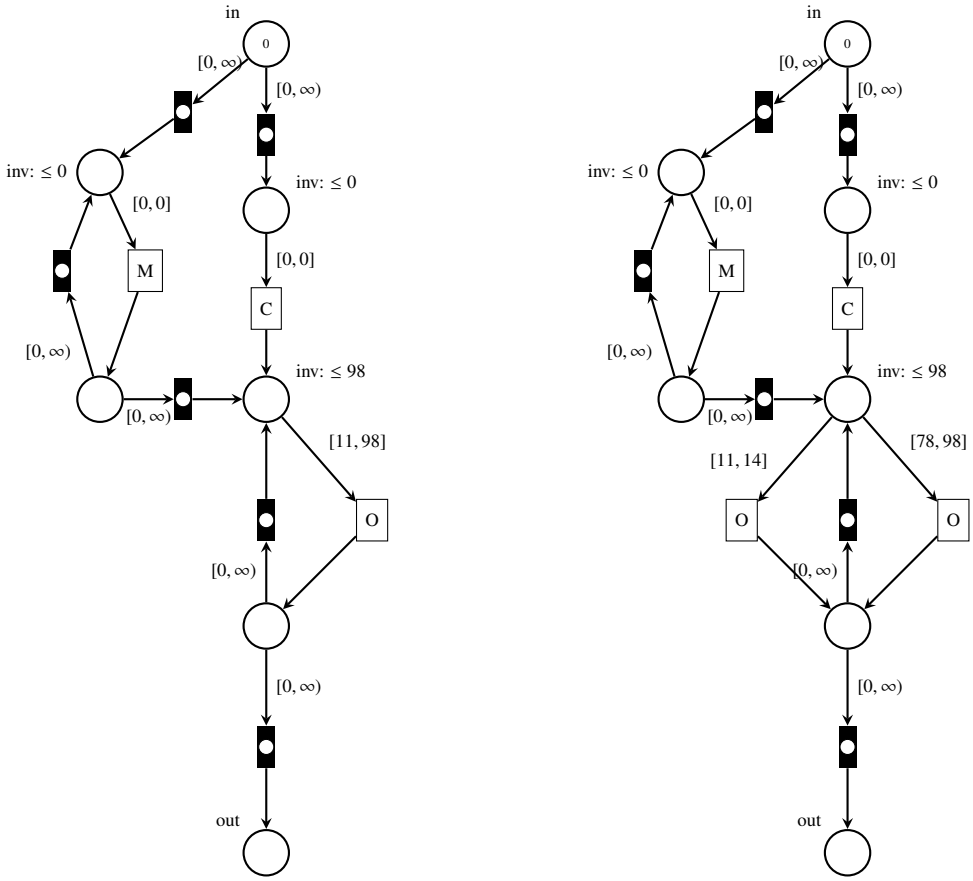
(b) *Petri net from ProM where the black dot represents a token and all τ-transitions are black*

Figure 13: *Models mined from Table 3*

The ETAWFNs presented in the remainder of our thesis, including Figure 14a, have been constructed such that they adhere to the constraints that we defined in our previous work [11, pp.26–27]. The constraints was defined as a prerequisite for any ETAWFN in order for it to be used in our distance measuring algorithm also developed as part of our previous work presented in [11]. In Figure 14a we have taken the minimum and maximum values of the observed O delays as the lower- and upperguard for O respectively. Say a new observation of O with 50 seconds delay is made as seen in Table 7a. The 50 seconds delay of this new O fits into the guard $[11, 98]$, but it does not cooperate our initial observations where we expect O to have a delay in the neighbourhood of $(11, 14)$, $(78, 86)$, or $(83, 85, 98)$ i.e. we have a false positive. Another method to determine the guard is to take the mean value of the initial observed O delays and offset it to get the upper- and lowerguard. However, applying this method to our example with any reasonable offset results in false negatives (and false positives) e.g. a 25 % offset produces the guard $[48, 80]$. In summary, the methods that combine the initial observed O delays into one guard exhibit symptoms of an underlying imprecision problem.

One's first inclination to solve the imprecision problem might be to simply split the O transition in the Petri net into three separate O transitions. However, this will introduce unecessary complexity thereby reducing what Aalst [9] refers to as a model's simplicity. We see this impact on the model's simplicity as a detriment since Aalst [9, p. 189] argues *"... the simplest model that can explain the behaviour seen in the log, is the best model"*. Furthermore, this solution only becomes a greater detriment to a model's simplicity as its size

and complexity increases. To explore a better method to combine the process map and the Petri net we use Figure 14b. Furthermore this adheres to a constraint we defined in [11], that all tau transitions must be urgent.



(a) *Poor solution: Min and max of O used in the guard* **(b)** *Better solution: Grouped values of O used in the guards*

Figure 14: *Sound ETAWFNs based on the Petri net in Figure 13b and annotated with time from Figure 13a where τ-transitions are urgent*

In the beginning of this section we mentioned that there was a pattern in the delay of the O events in Table 3. There were some fast O events $(11, 14)$ and some slow O events $(78, 83, 85, 86, 98)$. It seems intuitive to exploit this pattern in our construction of the ETAWFN in order to improve time discrimination. In Figure 14b we have constructed an ETAWFN with an additional O transition such that we can have one O transition for fast Os $[11, 14]$ and another for slow Os $[78, 98]$. This ETWAFN is indeed better at discriminating O events with regards to their delays e.g. it correctly dismisses the O event in Table 7a. The method used to construct the ETAWFN in Figure 14b is more precise than that of the ETAWFN in Figure 14a, but it may still be refined further.

In Table 7b we have a new observation of an O event but now with a delay of 87 seconds. This new O event is consistent with our initial observations and fits into the guard $[78, 98]$. However, if we consider its preceding event in addition to its delay we see a flaw in the ETAWFN in Figure 14b. The initial observations of the sequence C followed by an O took 11 or 14 seconds; hence, the new O most likely is erroneous and should be recognised as such.

Act	Order number	Delay (sec)
C	12	8
O	12	58

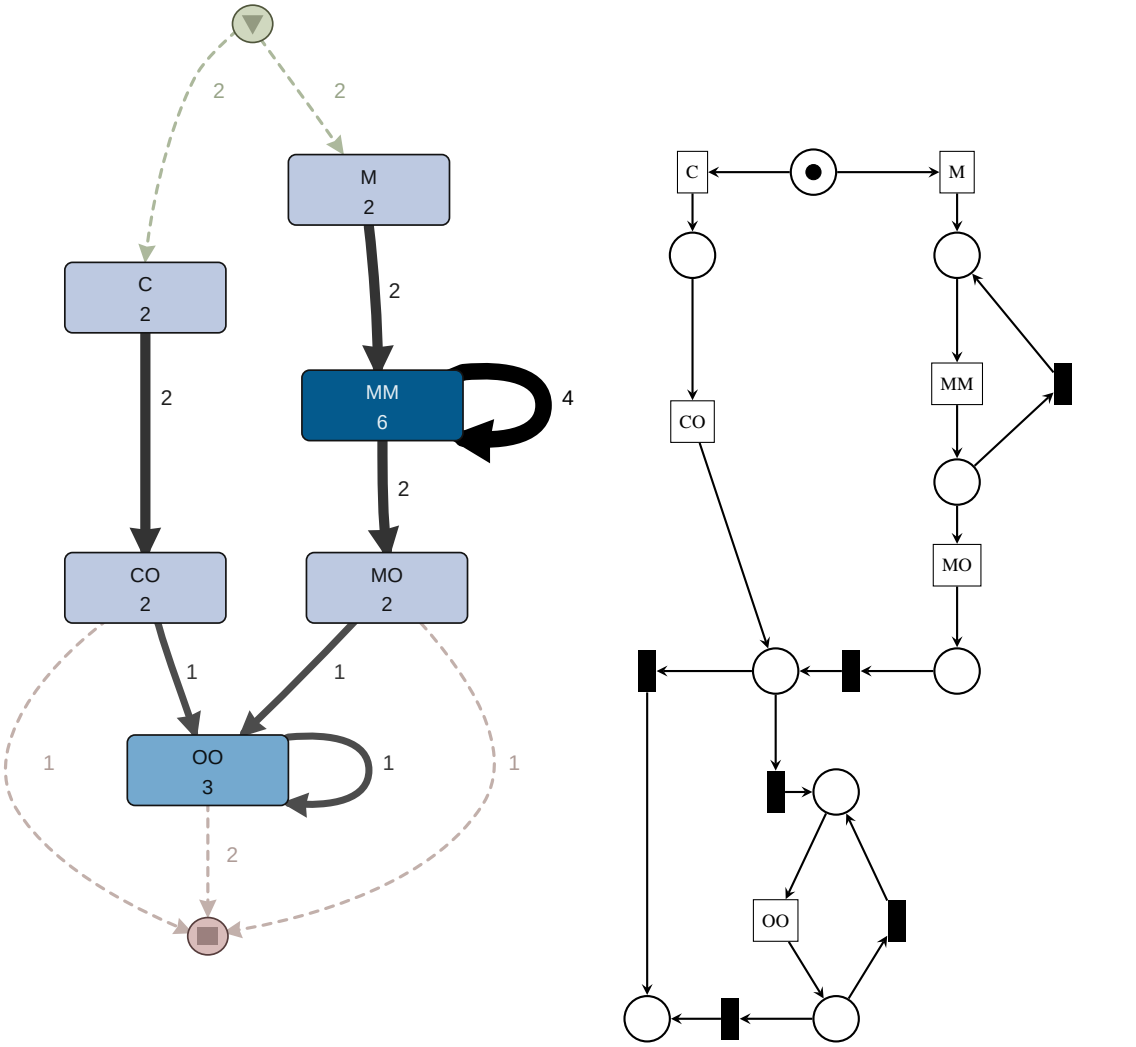
(a) *O event with a delay of 50 seconds*

Act	Order number	Delay (sec)
C	15	7
O	15	94

(b) *O event with a delay of 87 seconds*

Table 7: *New Observations of O events*

In order to also discriminate based on the order in which events occur we propose to concatenate the event types. As an example take the trace in Table 7a. After concatenating the events in this trace it will now begin with a C followed by a CO. In Appendix A Table 15 we have applied this method to the log in Table 3 which we have used so far. Then, by mining the log new log in Table 15 with the concatenated events in Disco and ProM we discover the models in Figures 15a and 15b respectively.



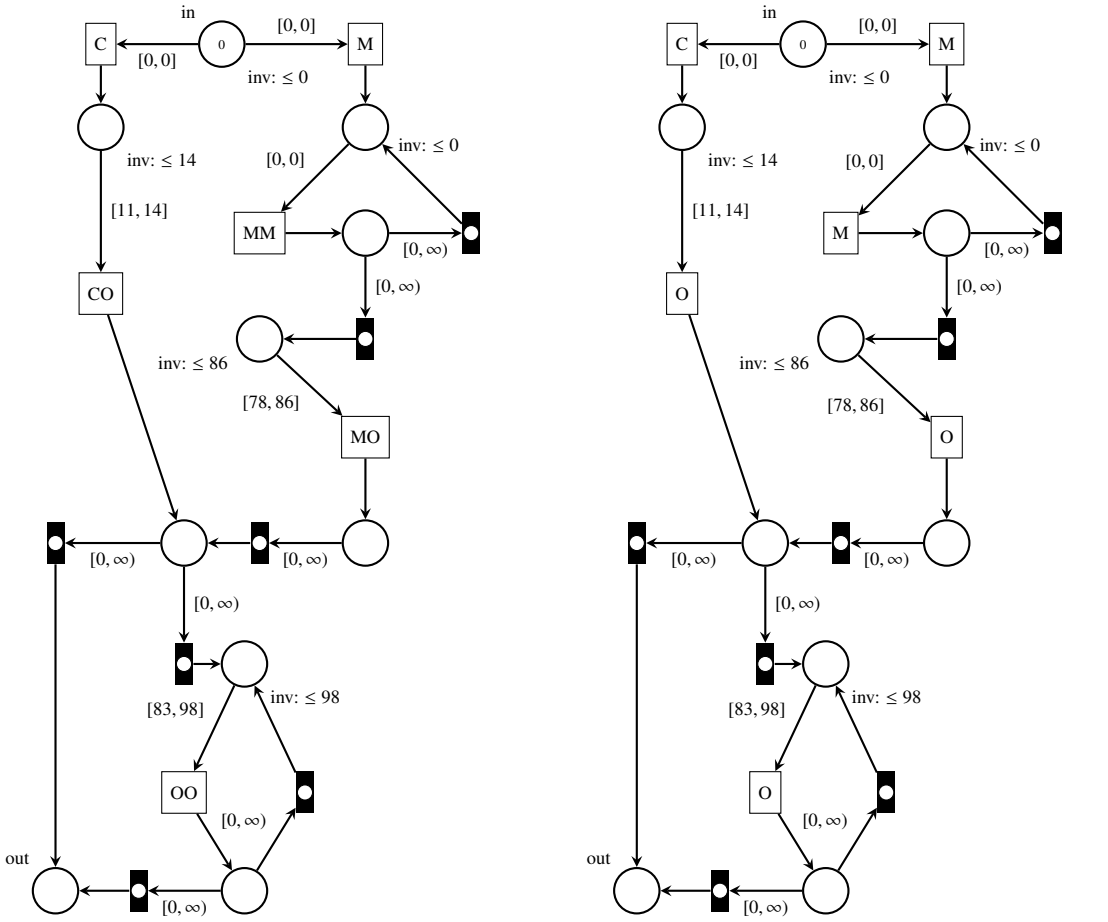
(a) *Process map from Disco*

(b) *Petri net from ProM where the black dot represents a token and all τ-transitions are black*

Figure 15: *Models mined from the log in Table 15 which contains the concatenated events from Table 3*

By comparing Figures 15a and 15b it is clear that the task of annotating the Petri net with time becomes a much simpler task since there now almost is a one-to-one mapping between the nodes and arcs in the process map, and the input arcs and transitions in the Petri net. The one exception is the MM node/transition. In the process map we have two arcs going into the MM node, but we only have one input arc into the MM transition in the Petri net. This is a problem in our example since M always occur with delay 0 but it is an early indication that the concatenating method alone may not be a complete solution to our time dilemma. However, exactly how well both the grouping and concatenating methods perform separately and in unison is something we shall return to later in the thesis, when we perform a set of experiments. By combining Figures 15a and 15b we construct the ETAWFN shown in Figure 16a.

As you can see in Figure 16a each transition, except for the two first, are labelled with the preceding event type and then its own event type. In our work we will "un-concatenate" the ETAWFN after it has been constructed such that we can replay traces in it without pre-processing said traces. When we un-concatenate Figure 16a we will get the ETAWFN in Figure 16b.



(a) *Concatenated ETAWFN*

(b) *Concatenated ETAWFN where original type names have been restored*

Figure 16: *Sound ETAWFNs based on the Petri net in Figure 15b and annotated with time from Figure 15a where τ-transitions are urgent*

In summary, we have suggested two methods for pre-processing an event log before a model is discovered from said log. The first method focused on improving the time discrimination with regards to each event by grouping the respective delays of an event. The second method aimed at concatenating events such that the order in which events occur is preserved after the discovery process thereby better discriminating events based on their order of occurrence.

11.2. Partitional Clustering

In order to avoid the time dilemma previously discussed in Section 11.1 when discovering a timed model, such as an ETAWFN, we propose to use clustering. In general, clustering is the activity of classyfying

patters (e.g. observations, or data items) into groups (clusters). Any sensible clustering process strives to produce clusters that exhibit a higher degree of internal pattern similarity than external pattern similarity. Jain et al. [36] distinguishes between two types of clustering: *Hierarchical* and *partitional*. A hierarchical clustering algorithm will produce a series of nested partitions, where as a partitional clustering algorithm yields partitions that optimise some clustering criterion. Since our aim is to partition a range of integers into disjoint intervals we will use partitional clustering. We clearly distinguish between the number partitioning problem [37, 38, 39] and partitional clustering since the former allows solutions where set elements have been rearranged. We define the problem of partitional clustering in context of our work in Definition 11.2.

Since the delays associated with the discrete-time events in a discrete-time event log may be repetitions of one another we shall focus our partitional clustering efforts on multisets. A given multiset may be partitioned in any various different combinations, where we will refer to the individual partitions of said multiset as *clustering instances*. In Definition 11.2 we define this term.

Definition 11.1 (Clustering instance)**.** A clustering instance $(A_1, A_2, \ldots, A_m)$ of a multiset A is a sequence of sub-multisets of A such that

a) $\bigcup_{i=1}^{m} A_i = A$, and

b) $A_i \cap A_j = \emptyset$, for all $i \neq j$.

In order to evaluate a given clustering instance we use the squared error criterion as it is argued by Jain et al. [36] to be "*the most intuitive and frequently used criterion function used in partitional clustering*". Given a clustering instance $\mathcal{L} = (A_1, \ldots, A_m)$ of a multiset A consisting of m clusters we define the squared error criterion function as seen in Equation (5). In Equation (5) let c_j be the centroid of cluster j which in our work is the mean value of cluster j.

$$e^2(\mathcal{L}) = \sum_{j=1}^{m} \sum_{x \in A_j} (x - c_j)^2 \tag{5}$$

From Equation (5) we get a value representing the summarised variance for each cluster which we seek to minimise. However, in order to compare the threshold value t to our squared error criterion we need a normalised value. For this purpose we first define Equation (6) and then Equation (7).

In Equation (6) we compute the summarised variance of a multiset A where x_i is the i$^{\text{th}}$ element in A. Additionally, we fix the centroid c to be the mean value of A.

$$e^2(A) = \sum_{i=1}^{|A|} (x_i - c)^2 \tag{6}$$

With the cluster and set variance defined we then define the *disposition* of a clustering instance $\mathcal{L}$ of a multiset A as seen in Equation (7). The disposition of a clustering instance produces a number between zero and one where a zero indicates no intra-cluster variance and a one indicates perfect intra-cluster variance.

$$\mathit{disposition}(\mathcal{L}, A) = \frac{e^2(A) - e^2(\mathcal{L})}{e^2(A)} \tag{7}$$

With the definition of a clustering instance and a way to evaluate a given clustering instance via its disposition we now define our partitional clustering problem as seen in Definition 11.2.

Definition 11.2 (Partitional clustering)**.** Given a multiset $A \subseteq \mathbb{N}^0$, and a threshold $t \in [0, 1]$, partition A into $A_1, A_2, \ldots, A_m$ clusters such that

- The disposition as defined in Equation (7) is greater than or equal to t while minimising m, and
- $|A_j| \geq \frac{|A|}{m} * 0.1$

In Definition 11.2 we chose to define a threshold value instead of the number of clusters to partition a set into. We did so since we want the patterns present in a set to determine the number of clusters and not a fixed parameter. Additionally, we also stipulate that m must be minimised since simple is always better in the context of process mining. Secondly, we also require that each cluster accounts for a reasonable amount of the total data items. We formulate this requirement in order to cope with what Aalst [9] refers to as noise. The term noise refers to rare and infrequent behaviour in the log and not logging errors. By this notion of noise we consider clusters that account for a relatively small amount of the total data items as *noise clusters*.

11.2.1 K-means Clustering

The k-means clustering algorithm as developed by Lloyd [40] is a well-known clustering algorithm due to its simplicity and its observed speed [41]. The algorithm takes as input a set $X = \{x_1, x_2, \ldots, x_n\}$ of points in $\mathbb{R}^d$. Then, by providing the algorithm with a seed of k centres $c_1, c_2, \ldots, c_k$ it will, according to Arthur and Vassilvitskii [41], partition X following steps one through three.

1. For each $i \in 1, \ldots, k$, set the cluster C_i to be the set of points in X that are closer to C_i than they are to c_j for all $j \neq i$.
2. For each $i \in 1, \ldots, k$, set c_i to be the center of mass of all points in C_i : $c_i = \frac{1}{|C_i|} \sum_{x_j \in c_i} x_j$.
3. Repeat steps 1 and 2 until c_i and C_i no longer change, at which point return the clusters C_i

In case two points are equally close to a point in X the tie is broken arbitrarily. If a cluster is empty at the end of step two it is eliminated and the algorithm continues as before. The steps of k-means function as stepwise refinement since each point in X is assigned to the centre it is closest to in conjunction with the fact that the centres stabilise with regards to the points they serve. This means that the clustering cost is reduced in each step and that no partitioning can be seen twice. Additonally, the k-means algorithm terminates in finite time since there are only finitely many k-clusterings [42].

According to Har-Peled and Sadri [42] the output of k-means is not guaranteed to be a global minimum and it can be arbitrarily bad compared to the optimal clustering. Furthermore, the answer returned by k-means and the number of steps it performs has been shown to depend on the initial choice of centres. Efficient polynomial time approximation schemes for both low and high dimensions have been developed to rectify the shortcomings of k-means. These, however, is argued by the authors to have been of little practical use since *"they are complicated and probably impractical because of large constants"*. Hence we shall stick to the basic k-means algorithm in our work.

In our work we shall apply the k-means algorithm on one dimensional integer arrays. Since this is the case we can rely on the k-means complexity given by Har-Peled and Sadri [42] who have proved an upper limit on the number of steps of k-means in one dimensional Euclidean space. This upper bound does not involve the number of clusters k but rather depends on the spread Δ in the input set. Thus, given a set of n points with spread Δ the upper bound of k-means is given by $O(n\Delta^2)$.

Using k-means alone does not consider two parts of Definition 11.2. I.e. the minimising of the amount of clusters, and the minimal size of each cluster. In the following, we explain how we handle the parts individually where the the minimising of clusters is handled through the use of k-means, and the check of the cluster sizes are handled after the k-means calculation.

11.2.2 From Clusters to Intervals

Using the k-means clustering algorithm we can now define an algorithmic solution to the partitional clustering problem as defined in Definition 11.2. The clustering instance returned by k-means is a local minimum which inherently satisfies the requirement that all elements in the input multiset is assigned to exactly one cluster. It may, however, not be the case that the disposition of the returned clustering instance is grater than or equal to the disposition threshold, nor is it necessarily the case that the number of clusters is minimised. These two concerns may be remedied in tandem by repeatedly running k-means until the disposition threshold is met beginning with $k = 2$, incrementing k for each k-means iteration. Following this incremental procedure the

clustering instance returned by k-means that satisfies the disposition threshold also minimises the number of clusters. Notice that the incremental procedure is guaranteed to always terminate when k is equal to the size of the input multiset, since the associated clustering instance always has a disposition of one. Finally, we need to remove any noise clusters from the clustering instance returned by the incremental k-means procedure.

Removing noise is a vital aspect of any discovery process. As part of the partitional clustering problem we explicitly require that noise clusters are excluded from a final solution. This, however, may not be enough to remove all noise from the input data. It may be the case that one or more clusters in the final clustering instance contains what we shall refer to as *noise items*. As an example take the fictional clustering instance $((5, 7, 11, 11), (41, 46, 48, 53, 55, 71), (175))$ where no noise filtering has been applied. The cluster (175) will be removed since it is a noise cluster. The cluster $(5, 7, 11, 11)$ has a low spread indicating no noise items, but this is not the case for $(41, 46, 48, 53, 55, 71)$. The 71 value seems intuitively as a noise item. To register the value as such, first need to relax condition a of Definition 11.2 as the union of all clusers now only has to be a subset of A, rather than equal to A, furthermore we introduce the theory of a box plot presented by Potter et al. [43], and illustrated in Figure 17, where given a set A of positive integers including zero:

Median is the value where 50 % of the data in A is below and above,
Lower Quartile ($Q1$) is the value where 25 % of data in A is below, and 75 % is above
Upper Quartile ($Q3$) is the value where 75 % of data in A is below, and 25 % is above
Outlier is any value $x \in A$ where $x \leq Q1 - 1.5 * (Q3 - Q1) \vee x \geq Q3 - 1.5 * (Q3 - Q1)$
Maximum is the largest value in A, which is not an Outlier, and
Minimum is the smallest value in A which is not an outlier.

By using the cluster $(41, 46, 48, 53, 55, 71)$, we get the box plot shown in Section 11.2.2 where we see that 71 has been noted as an outlier, and maximum is set to 55, as it is the largest value in the cluster, hence we see that noise element is caught.

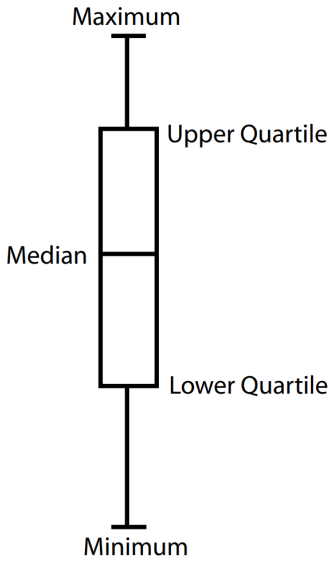


Figure 17: *Illustration of a boxplot, [43]*

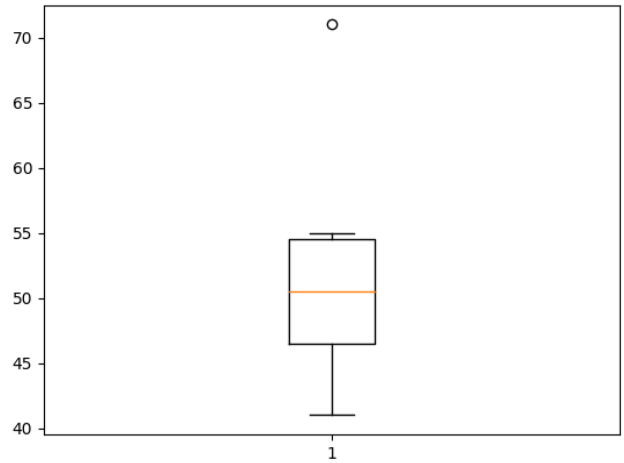


Figure 18: *Boxplot of the cluster* $(41, 46, 48, 53, 55, 71)$ *with whiskers reach of* $1.5 \times IQR$

Once each cluster has been filtered for noise, we remove all clusters that do not adhere to Definition 11.2, i.e. where the cluster consists of too few elements. Then each remaining cluster can be converted to an interval $[lower, upper]$ simply by taking the minimum and maximum values in the cluster. As an example take the cluster $(41, 46, 48, 53, 55, 71)$ which will produce the interval $[41, 55]$. We summarise the process of

determining the appropriate intervals for a data set in the two following steps:

i Use the incremental k-means process to find a solution to the partitional clustering problem as defined in Definition 11.2

ii Filter out noise items in each cluster and determine each cluster's interval

These two steps results in the makeIntervals(A, t, m, w) function shown in Algorithm 2.

Algorithm 2: makeIntervals pseudocode

```
1  function makeIntervals(A, t)
       input  : A multiset A, and a threshold value t
       output : A set of intervals
2      disp := 0
3      n := 2
4      L := A
5      intervals := {}
6      while disp < t do
7          L := calculate k-means clustering for A with n clusters
8          disp := disposition(L, A)
9          n := n + 1
10     forall c ∈ L do
11         let[Q1, Q3] := calculate boxplot for elements in the cluster c
12         forall x ∈ c do
13             if x ≤ Q1 − 1.5 ∗ (Q3 − Q1) ∨ x ≥ Q3 + 1.5 ∗ (Q3 − Q1) then
14                 remove x from c
15     forall c ∈ L do
16         if |c| < |A|/n ∗ 0.1 then
17             remove c from L
18     forall c ∈ L do
19         let interval be a pair (min(c), max(c))
20         add interval to intervals
21     return intervals
```

12. Distance Algorithm

In Sections 13.4 and 13.6 we shall use a distance algorithm devised in our previous work [11] in order to evaluate the two methods proposed in Section 11.1 to solve the time dilemma and later in Section 13.6 examine the effectiveness of the distance algorithm. As part of our previous work we implemented the distance algorithm in c++ as part of VerifyDTAPN [44] which is the verification engine used in TAPAAL. Furthermore, we also implemented automatic loading of discrete-time event logs from eXtensible Markup Language (XML) documents. In the following section we focus on providing a formal description of the distance algorithm.

12.1. Pseudocode

This distance algorithm will compute the minimal distance between a given ETAWFN and a discrete-time event trace (DTET). As a prelude to the discussion of the individual functions which constitute our distance

algorithm we first examine on a logical level how the distance measuring algorithm computes the distance for a given ETAWFN and DTET. The distance computed for a DTET is the accumulated distance for each timed action in the DTET. Each timed action (a, d) is processed through three unique steps which we denote as: $M \xrightarrow{\tau}{}^{*} M' \overset{d}{\Rightarrow} M'' \overset{a}{\Rightarrow} M'''$. According to these three steps a timed action is processed as follows:

i $M \xrightarrow{\tau}{}^{*} M'$ entails finding a marking M' from M such that a transition with label a can be fired and the resulting distance of steps *ii* and *iii* is minimised.

ii Once M' is found, the appropriate delay is performed, $M' \overset{d}{\Rightarrow} M''$. We deliberately have not used the notation $M' \xrightarrow{d} M''$ since it may not be possible to delay d time units in M'' hence we must do a cheating delay signified by $\overset{d}{\Rightarrow}$. This cheating delay is always performed such that the resulting distance from delaying d is minimised.

iii Subsequently to the cheating delay we perform a cheating firing, $M'' \overset{a}{\Rightarrow} M'''$. As with the cheating delay, performing a cheating firing is also done such that the resulting distance is minimised.

The timed action (a, d) has now been replayed such that the distance is minimal. These three steps are repeated for every timed action until the entire DTET have been replayed and the minimal distance for the DTET has been computed.

With this stepwise reasoning behind our distance algorithm we now move on to discuss the individual functions which constitute the distance algorithm; presented in Algorithms 3 through 8. Below we provide a brief summarisation of each function in order to give an overview of each function's purpose and subroutines.

Algorithm 3 Iterates over each timed action (a, d) of a DTET π, applying step i as a BFS followed by the subroutine calculateDistance(M, d, a) (Algorithm 4) and returns the distance between n and π.

Algorithm 4 Performs step *ii* followed by step *iii* using the subroutines delayDistance(M, d) (Algorithm 5) and firingDistance(M, a) (Algorithm 6) respectively. The function returns the penalty for cheat delaying d followed by cheat firing a and the resulting marking.

Algorithm 5 Performs a cheating delay d in a marking M such that the resulting penalty is minimised i.e. step *ii*. The marking produced by cheat delaying d is returned together with the penalty for reaching it.

Algorithm 6 Iterates over all transitions t where $L(t) = a$ applying step *iii* and returns the penalty for cheat firing the cheapest t, calculated by the subroutine enableTransition(M, t) (Algorithm 7), and the resulting marking.

Algorithm 7 Determines and performs the necessary changes in a marking M to enable a transition t. This function spawns tokens if needed but delegates the task of modifying token ages to the subroutine modifyTokenAges(X, M, a, b, n) (Algorithm 8). The marking in which t is enabled is returned together with the minimal penalty for enabling t.

Algorithm 8 Modifies the age of n tokens by iterating over a multiset of tokens X from a place p. The function determines the cheapest age modification to make for a given token x using a lowerguard a and upperguard b. The marking resulting from the neccesary age modifications, such that the penalty is minimised, is returned together with the penalty for reaching this marking.

Algorithm 3: distance function pseudocode

```
1  function distance(N, M_0, π)
       input  : A TAPN N with its initial marking M_0, and a discrete-time event trace π
       output : Distance between N and π
2      penalty := 0
3      M := M_0
4      while π ≠ ∅ do
5          Dequeue (a, d) from π
6          Run BFS on τ-transitions from M and fire M -τ*-> M' s.t. M' minimises
             calculateDistance(M', d, a) and has the shortest τ-sequence
8          let (M'', penalty_dist) := calculateDistance(M', d, a)
10         penalty := penalty + penalty_dist
11         M := M''
12     return penalty
```

Algorithm 3 computes the distance between a given ETAWFN in its initial marking M_0 and a DTET π. The minimal distance for each timed action is computed through the while loop on line four. After having dequed the next timed action (a, d) we perform a Breadth-First Search (BFS) for a sequence of τ-transitions for a marking M' that has some t such that $L(t) = a$ i.e. we perform the first step; $M \xrightarrow{\tau}{}^{*} M'$. It is important to notice that since we allow for duplicate transitions (transitions with the same label) it may be the case that we find multiple M' markings with the same τ sequence length. Because of this we also stipulate that the M' we decide to continue with, minimises the distance.

Algorithm 4: calculateDistance function pseudocode

```
1  function calculateDistance(M, d, a)
       input  : A marking M with a delay d and a label a
       output : The penalty for delaying d followed by firing a in M
2      let (M', pen_1) := delayDistance(M, d)
3      let (M'', pen_2) := firingDistance(M', a)
4      return (M'', pen_1 + pen_2)
```

For each marking found through the BFS in Algorithm 3 we calculate their respective distance using Algorithm 4 as shown above. Algorithm 4 computes the two final steps, $M' \overset{d}{\Rightarrow} M'' \overset{a}{\Rightarrow} M'''$, as *delayDistance()* and *firingDistance()* respectively. Algorithm 4 returns the two distances added together with the marking M'' reached by cheat delaying d followed by cheat firing a. Notice that the marking reached in Algorithm 4, M'',

is equivalent to the marking, M''', reached by the final logical step.

Algorithm 5: delayDistance function pseudocode

```
1  function delayDistance(M, d)
       input  : A Marking M and a discrete-time delay d
       output : A marking M' resulting from delaying d time units in M and the penalty for
                performing said delay
2      penalty := 0
3      M' := M
4      forall p ∈ P do
5          let [0, b] := I(p)
6          M'(p) = {min(x + d, b) | x ∈ M(p)}
7          forall x ∈ M(p) do
8              if x + d > b then
9                  penalty := penalty + (x + d − b)
10     forall t ∈ T do
11         if M --t--> ∧ t is urgent then
12             penalty := penalty + d;
13     return(M', penalty)
```

In Algorithm 5 we perform the cheating delay for d. First, we delay the minimal amount of time units for all places p in the current marking through the forall loop on line four. In this forall loop we decide and perform the actual delay on line six and then we penalise appropriately through the forall loop on line seven. In the forall loop on line 10 we penalise for any enabled urgent transitions.

Algorithm 6: firingDistance function pseudocode

```
1  function firingDistance(M, a)
       input  : A marking M and a label a
       output : A marking M' resulting from firing t in M s.t. L(t) = a and the penalty for firing t
2      penalty := ∞
3      M' := M
4      forall t ∈ T s.t. L(t) = a do
5          let (M_t, penalty_t) := enableTransition(M, t))
6          if penalty_t < penalty then
7              penalty := penalty_t
8              M' := M_t --t-->
9      return (M', penalty)
```

In Algorithm 6 we perform the cheat firing of a transition with label a. In the forall loop on line four we iterate over all transitions with label a in order to find the one candidate that has the lowest penalty. The penalty associated with cheat firing a candidate is calculated by the subroutine $enableTransition(M, t)$. If this calculated penalty is indeed lower than the current best candidate we update the penalty and fire the candidate transition, lines six through eight. Notice that we on line eight use the notation $M_t \xrightarrow{t}$ and not $M_t \overset{t}{\Rightarrow}$. We do this since the candidate transition t is enabled in the marking M_t returned by $enableTransition(M, t)$.

We have divided the task of enabling a transition into two subtasks: Decide the necessary token spawns and time modifications, and modify ages of tokens. Algorithm 7 determines the necessary changes that are

needed to enable t which entails spawning tokens and/or modifying the ages of tokens. The actual time modifications of token ages are then performed by Algorithm 8.

Algorithm 7: enableTransition function pseudocode

```
1  function enableTransition(M, t)
       input  : A marking M and transition t
       output : The penalty for enabling t in M and the resulting marking M' for doing so
2      penalty := 0
3      M' := M
4      forall p ∈ •t do
5          C1 := |{x ∈ M(p) | x ∈ g(p, t)}|
6          C2 := |M(p)|
7          C3 := w(p, t)
8          let[a, b] := g(p, t)
9          if C1 < C3 then
10             if C3 ≤ C2 then
11                 let (M_mod, penalty_mod) := modifyTokenAges(M(p), a, b, (C3 − C1))
12                 penalty := penalty + penalty_mod
13                 M' := M_mod
14             else
15                 penalty := penalty + C3 − C2
                                        C3 − C2 tokens
16                 M'(p) := M(p) ∪ {a, ..., a}
17                 let (M_mod, penalty_mod) := modifyTokenAges(M'(p), a, b, (C3 − C1))
18                 penalty := penalty + penalty_mod
19                 M' := M_mod
20     return (M', penalty)
```

Algorithm 7 iterates over all the pre-places of t in order to determine what changes that are needed to enable t. The necessary changes for a place p is decided based on the three variables C_1, C_2, and C_3.

C_1**:** The number of tokens in p that satisfy the guard of the input arc from p to t.

C_2**:** The total number of tokens in p.

C_3**:** The weight of the input arc from p to t.

Additionally, we set a and b to be the lower- and upperguard respectively. If there are less than $w(p,t)$ tokens in p that have an age in $g(p,t)$, i.e. $C_1 < C_3$, we enter the body of the if statement on line nine and now face two possible scenarios. In the first scenario, $C_3 \leq C_2$, the number of tokens in place p is at least as many as we need according to C_3 but not not enough of them currently satisfy $g(p,t)$; hence, we only need to modify the age of $C_3 - C_1$ of the existing tokens in p. In the second scenario, $C_3 > C_2$, there is not enough tokens in p to satisfy $g(p,t)$; hence, we spawn $C_3 - C_2$ tokens and modify the ages of $C_3 - C_1$ required by C_3. We perform this computation for each pre-place of t and continuously update M' and *penalty* which the function also returns.

In [11] we impose a constaint of having to inhibitor arcs in the net, and aad we not imposed this constraint, then Algorithm 7 would also have to handle any inhibitor arcs going into t. This is a feature that if included in

our pseudocode would utilise the type function *type()* which was presented as part of Definition 10.2.

Algorithm 8: modifyTokenAges function pseudocode

```
1  function modifyTokenAges(X, M, a, b, n)
       input  : A multiset of tokens X present in M, a lower- and upperguard a, b, and number of
                tokens to modify n
       output : The penalty for fitting the age of n tokens to be in the range [a, b] and the resulting
                marking M'
2      penalty := 0
3      M' := M
4      let candidates be a list[n] of pairs(token, diff := ∞)
5      forall x ∈ X do
6          i := 0
7          for i < n do
8              let (t_i, diff_i) := candidates[i]
9              if |diff_i| > x − b ∧ x − b > 0 then
10                 remove last element in candidates
11                 insert pair[x, b − x] into candidates[i]
12                 break for
13             else if |diff_i| > a − x ∧ a − x > 0 then
14                 remove last element in candidates
15                 insert pair[x, a − x] into candidates[i]
16                 break for
17             i := i + 1
18     forall e ∈ candidates do
19         let (t_e, diff_e) := e
20         modify x in M' s.t. x = t_e to x := x + diff_e
21         penalty := penalty + |diff_e|
22     return (M', penalty)
```

Algorithm 8 modifies the age of n tokens from the multiset of tokens X. We manage which tokens to modify, and how, using the list of pairs *candidates* declared on line four. We iterate over all tokens in X on line five and determine if each individual token x should be considered a viable candidate using the loop on line seven. In this loop we iterate over all current candidates i and add x as a new one if one of the two conditions on line nine or 13 is met. These two statements check if modifying x with respect to the upper- or lower guard, b and a respectively, yields a lower penalty than the *diff* of i. We allow for both positive and negative values for *diff* such that when we begin to modify M' we can simply add the *diff* of a given candidate to its corresponding x.

12.2. Example

Now that we have presented the distance algorithm as pseudocode we follow it up with a concrete example using the discrete-time event trace in Table 8 and the extended timed-arc workflow net (ETAWFN) in Figure 19. Our distance algorithm will replay the sequence of events in the log beginning with the first M event and with the ETAWFN in its initial marking as shown in Figure 19.

First step i is performed where the τ-transitions in the postset of in is explored producing two separate markings $\{(P1, 0)\}$, and $\{(P2, 0)\}$. In the marking $\{(P1, 0)\}$ we find a transition labelled M so we stop the τ

BFS. We then delay zero times units and fire the M transition since it is enabled, producing the marking $\{(P3, 0)\}$. The next M event is replayed following steps *i* through *iii* where we produce a token in P1 of age zero followed by a delay of zero time units and then the transition M is fired producing the marking $\{(P3, 0)\}$. So far we have not penalised any of the events.

When performing the τ BFS for the seubsequent C event we will not find a marking with a transition labelled C. According to Algorithm 7 line 16 we must now spawn a token in the preplace of C and if necessary cheat delay such that we enable C. This behaviour however exposes a flaw not in the pseudocode but in our implementation. Currently in the implementation we will not spawn any tokens, but only penalise a fixed value and move on to the next timed action in the trace. In our example we add a penalty of 100 and move on to the next O event. Consequently we replay the O event from the marking $\{(P3, 0)\}$. Through the τ BFS we find that the marking $\{(P4, 0)\}$ includes transitions labelled O. In this marking we delay 10 time units as this does not violate the invariant producing the marking $\{(P4, 10)\}$. However, neither of the O transitions are enabled in this marking; hence, we cheat delay 1 sec, penalise accordingly and produce the marking $\{(P4, 11)\}$. We then complete the replay of the O event by firing the now enabled O transition.

Finally we need to replay the I event. This is however not possible since there are no transitions labelled I. We therefore penalise a fixed value which in our case is 1000. The trace has now been replayed completely with a penalty of 1101.

Type	Location	Item	Quantity	Worker	Order number	Delay	Penalty
M	845-05-03-B	43	-33	avr	45	0	0
M	849-05-01-B	43	33	avr	45	0	0
C	401-01-03-B	17	4	avr	45	5	100
O	401-01-03-B	17	-4	avr	45	15	101
I	808-03-11-A	32	32	avr	45	96	1101

Table 8: *A fictional trace where the action types corresponds to those in Table 1*

13. Experiments

In this section we present four different experiments which we have conducted, each relating to different parts of the work we have conducted in this thesis, and in our previous work in [11]. Before presenting each experiment we, we cover how we for most of the experiments used a specific division of the data found in the provided logs.

13.1. Methodology

A common characteristic between our experiments is that they all aim at learning something about the behaviour represented in an event log. For our first experiment in Section 13.3 the aim is to show the impact of one or more swaps has, where as the subsequent experiments aim towards learning what the behaviour represented in the log is and how it changes. Due to the difference in aim between our first experiment and the learning experiments we shall apply slightly different methodology to the former and to the ladder.

For our first experiment we will use one event log where no swaps have been performed. Using one event log suffices since we can then modify it with swaps and compare the man hours in the event log before and after swaps.

Our learning experiments resemble the problem of supervised learning [45]. Hence we need two data sets; training data to produce our models, and test data to show how well the produced model fits on unknown data. Through research we found different suggestion to the division of the data, but we will follow the Pareto

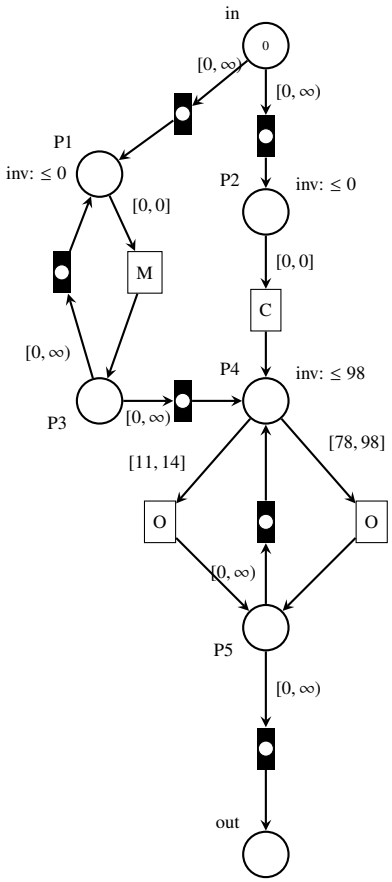


Figure 19: *The same ETAWFN previously shown in Figure 14b but all places have been labelled*

principal [46] otherwise known as the 80/20 rule, where we use 80 % of the data for training, and 20 % of the data for testing. The data could be split on a few different properties such as the time, traces, or on the items involved. As the purpose of our learning experiments is to examine the behaviour captured in the data we shall split on traces.

13.2. Data

In Section 5 we presented our case study of the business Av Form. As part of this case study we have available to us data logs stretching as far back as the 14^{th} if February 2018. In Table 9 we succinctly show the time period that we have data for and how we created three logs from the data. The first log L1 accounts for three months of log events, L2 also accounts for three months, and the last log L3 contains event data for four days due to an approaching thesis deadline/time constraints with regards to our thesis. We decided to let L1 and L2 account for three months as this time interval corresponds to a quarter of a year and thereby fits well with the notion of quarterly business reports.

	Type	Location	Item	Quantity	Worker	Order number	Time	
L1	I	859-05-03-B	47	31	w1	5	14-02-2018 08:19:23	
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
	I	147-01-03-B	12	02	w1	108	20-02-2018 08:04:31	L2
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
	C	856-08-05-B	15	41	w2	389	15-05-2018 08:01:01	
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
	O	785-03-05-B	26	15	w3	457	18-05-2018 08:04:59	
	Swap	758-08-02-B	54	11	w1	524	22-05-2018 08:00:14	L3
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
	Swap	856-04-03-B	55	23	w1	600	25-05-2018 08:02:35	

Table 9: *All data available during the experiments distributed between three logs: L1, L2, L3*

13.3. Evaluating Swap Effectivenes

To evaluate the swap effectiveness, we look at a configuration of the warehouse at Av Form, prior to providing our implementation. We base the calculation of the cost function presented in Equation (2) on page 14 on the three months log, L1, presented in Section 13.2, and we look at the logged actions over the same three months to see how the actual work performed in the warehouse during that period affects the result.

For this evaluation we first used the item average as defined in Definition 7.1, where no discounting was used. This choice was made primarily because the amount of times each location was accessed was low, hence we did not want the infrequent access to affect the results gathered. However, because of the results we got, we have chosen to also include variations of the Expenditure Definition 7.3, which we introduce in the following.

Furthermore, we picked the swap candidates based on a single calculation of the costs, as this correlates to how the work has been executed at Av Form.

Lastly, we calculated the estimates of time used to travel between each location, using the time estimates presented in Table 6 on page 18, where we determined how the worker would travel through the warehouse, by looking at a blueprint annotated with their s-routing structure.

To calculate the estimates of how much time has been spend in total for the three months of log, we calculate the accumulated time used for each event, by the following approach for all events in each trace using the time estimates presented in Section 8.2 on page 15.

1. Set the location of the first event as source destination,
2. Add the cost of travelling from source location to the location associated with the event,
3. Add the cost associated with the event to the total cost,
4. Set the source location to the location for the current event,
5. Get the next event in the trace and go back to item 2

We first follow these steps to replay all replenish tasks found in the translog, prior to performing swaps. Then we replay the same log where we have swapped a set of locations as explained in the above, and we added the swapcost associated with swapping the locations.

As mentioned, our initial test with the use of Equation (2) yielded some unexpected results, hence we decided to look at possible changes to the calculation of expenditure (Definition 7.3). Where we will first introduce two new definitions; *item_access* and *log_duration* in Definitions 13.1 and 13.2 respectively, where

item_access is the amount of times an item has been accessed in a log, and *log_duration* is a calculation of the duration of a log where seconds are scaled to days.

Definition 13.1 (Item Access)**.** Given a discrete-time event log Π of the form as show in Equation (1), we define the amount of times an item $\bar{i}$ has been accessed as:

$$item\_access(\bar{i}) = \sum_{\substack{j=1 \\ t_j=RO \\ \bar{i}=i_j}}^{n} 1$$

Definition 13.2 (Log Duration)**.** Given a discrete-time event log Π of the form as show in Equation (1), we define duration of a log as:

$$log\_duration(\bar{i}) = \frac{d_n}{60 * 60 * 24}$$

Given these two definitions, we extend on Definition 7.3 for expenditure in two ways, the first is Definition 13.3 where we include the amount of times each item has been accessed, without considering the duration of the log, and the second Definition 13.4 where less frequently accessed items will weight less than more frequently accessed items.

Definition 13.3 (Expenditure and item access)**.** Given a warehouse $WH = (L, I, Dist, Lift, Swapable)$ in a configuration C, we define the effect of swapping locations l_1 and l_2 as:

$$expenditure(l_1, l_2) = \left(\sum_{i \in l_2} Lift(l_1) * \frac{item\_access(\bar{i})}{\frac{Config(l_2)(i)}{item\_avg(i)}} + \sum_{i \in l_1} Lift(l_2) * \frac{item\_access(\bar{i})}{\frac{Config(l_1)(i)}{item\_avg(i)}} \right) - \left(\sum_{i \in l_2} Lift(l_2) * \frac{item\_access(\bar{i})}{\frac{Config(l_2)(i)}{item\_avg(i)}} + \sum_{i \in l_1} Lift(l_1) * \frac{item\_access(\bar{i})}{\frac{Config(l_1)(i)}{item\_avg(i)}} \right)$$

Definition 13.4 (Expenditure, item access, and log days)**.** Given a warehouse $WH = (L, I, Dist, Lift, Swapable)$ in a configuration C, we define the effect of swapping locations l_1 and l_2 as:

$$expenditure(l_1, l_2) = \left(\sum_{i \in l_2} Lift(l_1) * \frac{Config(l_2)(i)}{item\_avg(i)} * \frac{item\_access(\bar{i})}{log\_duration(\bar{i})} + \sum_{i \in l_1} Lift(l_2) * \frac{Config(l_1)(i)}{item\_avg(i)} * \frac{item\_access(\bar{i})}{log\_duration(\bar{i})} \right) - \left(\sum_{i \in l_2} Lift(l_2) * \frac{Config(l_2)(i)}{item\_avg(i)} * \frac{item\_access(\bar{i})}{log\_duration(\bar{i})} + \sum_{i \in l_1} Lift(l_1) * \frac{Config(l_1)(i)}{item\_avg(i)} * \frac{item\_access(\bar{i})}{log\_duration(\bar{i})} \right)$$

With the two Definitions 13.3 and 13.4, we calculated new swap suggestions, applied these swaps and calculated the estimated savings as shown in Table 10, where we included the savings both with and without the associated swapcosts.

	No Discounting	Item access	Item access and log duration
Swap suggestions	5	4	—
Estimate before swaps	426,834 sec	426,834 sec	—
Estimate after swaps	427,482 sec	427,268 sec	—
Difference	648 sec	434 sec	—
Swapcost	762 sec	538 sec	—
Difference without swapcost	-114 sec	-104 sec	—

Table 10: *Estimated time durations over replayed logs*

The results are seen in Table 10, there the third test did not yield any results, this is due to the strictness of the implementation, where it is calculated that no swaps could be performed, which would result in a decrease over the three months period. For the first approach, many swap candidates were suggested, so we chose to only look at the top 5 best candidates, whereas for the second approach, only four candidates where suggested. Looking at the difference for the two approaches (the first highlighted row) we see that both add to the time, rather than reducing the time.

If we however consider the cost function in Equation (2) on page 14, we include the cost of swapping two locations. We consider that this may not be relevant, as the workers are only expected to perform swaps, when they have no other preceding tasks, i.e. they perform the swap when they would be otherwise idle, and therefore performing the swap does not add to the cost. This is considered in the last highlighted row in Table 10 where we removed the swapcosts, and find that we do have a reduction in the estimated time for performing all events in a log.

13.4. Evaluating Our Time Discrimination Methods

In Section 11 we presented the time dilemma occurring under the discovery process. As a solution to the time dilemma we suggested two methods: Clustering of delays to improve time discrimination, and concatenation of event types to improve event discrimination. In order to evaluate the effectiveness of these methods we use our Distance algorithm previously discussed in Section 12. For the purposes of this experiment we will use the L2 log which we split into training and test data. From the 1970 traces which are present in the L2 log we then have 1576 training traces and 393 test traces.

In order to discover models based on the training data we use a combination of the tools Disco, ProM, and TAPAAL. Most noteworthy of these tools is ProM where we use the inductive miner algorithm [17] with a noise threshold of 0.10. We tried to use ProM with a noise threshold of 0.20 but found the resulting Petri net too restrictive and also with a noise threshold of zero but found the resulting Petri net too general. In Algorithm 2 we presented the makeIntervals(A, t) algorithm to create intervals from a data set. In this experiment we set the makeIntervals(A, t) parameters $t = 0.8$, and A as a list of delays for some event type.

Distance evaluation

To give a broad overview of the methods we shall compare them individually and to the method we used in our previous work which we refer to as the old method. Furthermore, we also evaluate the result of combining the concatenation and clustering methods. The methods we evaluate are briefly summarised as follows:

Old: First discover a process map and a Petri net. Annotate the Petri net with the minimum and maximum delay values as seen in the process map. This follows the same process as the one described in relation to Figure 14a.

Concatenating: First preprocess the event log such that all events are concatenated. Then, discover the process map and Petri net, and combine these following the process used to generate Figure 16.

Clustering: Preprocess the event log by executing Algorithm 2 makeIntervals(A, t). Then, only discover a Petri net and, by following the same procedure used to generate Figure 14b, annotate it with intervals.

	distance zero traces
Old	99.2 %
Concatenating	99.2 %
Clustering	32.3 %
Combined	42.7 %

Table 11: *The percentage of test traces with distance zero for the respective preprocessing methods*

Combined: Preprocess the event log with concatenation followed by clustering. Then, follow the annotation process of the clustering method.

In Table 11 and Figure 20 we compare the distances of the four methods. Our first point of comparison is the percentage of zero traces as seen in Table 11. Both the old and concatenating methods are incapable of discriminating based on time since their intervals are so broad; hence, almost all of the test traces has a distance of zero. Furthermore, the concatenating method did not improve event discrimination, since the distance algorithm will penalise a fixed amount if an event cannot be replayed without spawning tokens or its label is unrecognisable.

The remaining two methods, clustering and combined, present major improvements with regards to time discrimination as is evident from their zero trace percentage. By comparing their percentage of zero traces it would seem that the clustering method performs better. However, we contribute the increase in zero traces to the combined method's ability to fine tune intervals to the sequence in which events occur. It has 15 different action types where as the clustering method only has four. To explain the consequence of this difference take the case of an I event (see table Table 1). In the combined method this I action type will be split into several specialised Is depending on what can be observed from the event log e.g. an I preceded by another I, or an I preceded by a C and so on. In the clustering method all of the I observations are put into the same basket and the intervals for the I event therefore become more general. The impact of this difference is clear when comparing the two methods as shown in Figure 20. In this chart we have ordered the traces for each method based on their distance to the training model and selected the top 25 % of traces with the highest distance.

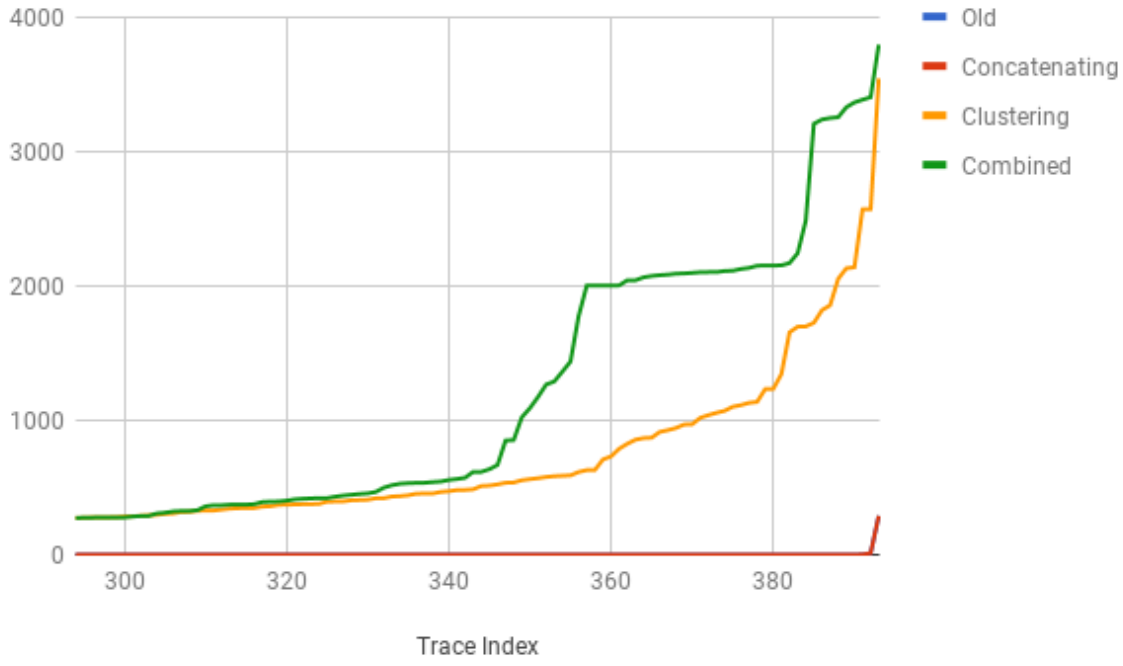


Figure 20: *The 25 % most penalised traces ordered by their distance*

The increase in precision of the combined method does however come with a cost to the complexity of the resulting ETAWFN. Recall that when the ETAWFN is un-concatenated we remove the preceding event from the action type e.g. an I preceded by another I denoted as I-I becomes I, an I preceded by an M M-I becomes I etc. This coupled with constraint three which we put on ETAWFNs in our previous work [11, p. 27] where any marking reachable from a marking m may not have two transition enabled simultaneously with the same label makes the un-cocatenating process of an ETAWFN cumbersome. Currently we have no automated way of un-cocatenating an ETAWFN; hence, we shall continue with the clustering method as it adequately discriminates events while being simple enough to use in a manual method.

13.5. Evaluating Workflows

For the workflow evaluation, we look only on select workers, for two reasons; there is a different amount of workers in the L2 and L3 logs, and the amount of rounds for each worker varies significantly. The distribution is seen in Tables 12 and 13.

Futhermore, for the generation of the models, we applied the clustering method for the time information in the logs, as this is the approach we concluded to use, during the previous experiment. Then we used the framework ProM to generate workflow nets using the specified 80 % of data used for training from logs L2, and L3.And we manually decorated the generated ProM models with the related cluster time information. The manual process was carried out in TAPAAL, where we also decorated some of the models with more τ-transitions, in order to make them applicable for our distance algorithm.

Lastly, as the logs originally only contain a small amount of action types, we decorated the types with more information, e.g. a transition with the label 'O-B-H' is read as; action type O, from zone B, and H specifying the height of a location. The possible zones are A,B, and C which are different regions of the

warehouse, and the possible heights are H and L, for high and low locations respectively, where locations up to 5 are low locations, and anything above, is a high location.

Worker	Orders
w1	968
w2	118
w3	817
w4	7
w5	14
w6	1
w7	1
w8	6
w9	2
w10	37

Table 12: *Order distribution in the L2 log*

Worker	Orders
w1	155
w2	124
w3	10
w4	3
w5	34
w6	1

Table 13: *Order distribution in L3 log*

From the two tables we see that workers w7 through w10 only are present in the L2 log, so these are disregarded as we seek workers present in both logs. We chose w1 and w2 as these workers have many orders in both logs, and for the third and last worker, we considered both w3, and w5 but decided on w3, as this worker has more total orders than w5.

In the following we present Petri nets showing workflows of the workers w1 through w3, where we first present the models generated on L2 followed by models generated on L3.

13.5.1 Workflow models for workers on log L2

For the first worker, the model has been split into three components shown in Figures 21 through 23, where the places and transitions with dotted lines are shared, i.e. the shared place P5 in Figure 21 is the same place as P5 in Figure 22. Furthermore, the events in the L2 log are only associated with replenish, and from our conversations with Logimatic, we know that they expect the action type RO to always be followed by the RI type. However, from observing the model, we can induce that this is indeed not the case as the model allows either type to be executed independent on the other. Furthermore, from observing Figure 22 we clearly see that action types have been divided into multiple partitions, i.e. from the multiple O-C-L actions depicted in Figure 22.

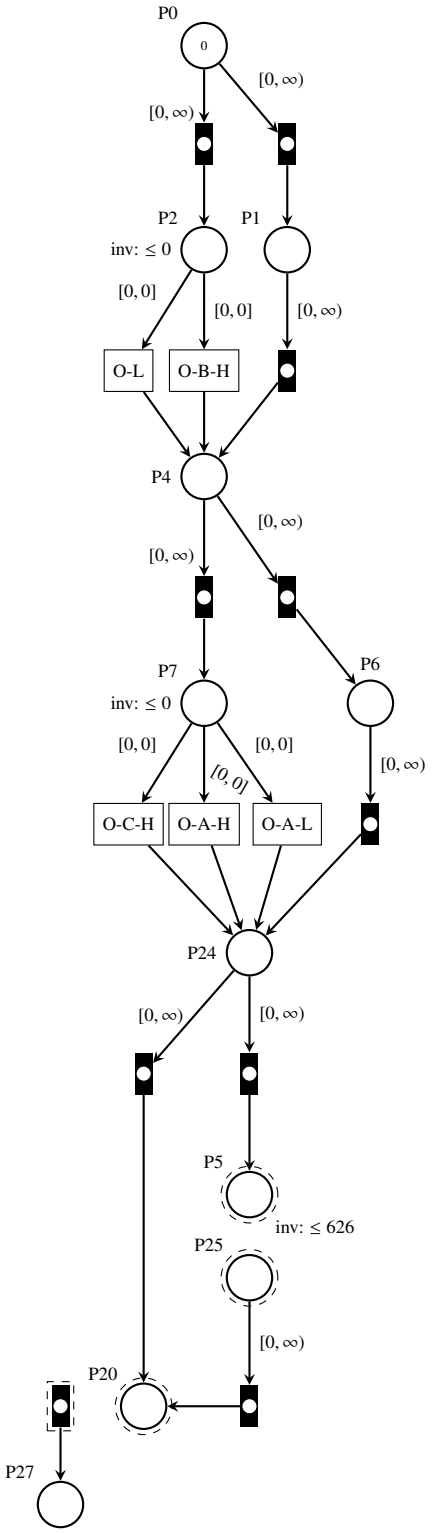


Figure 21: *Component 1 of workflow for w1 using the L2 log*

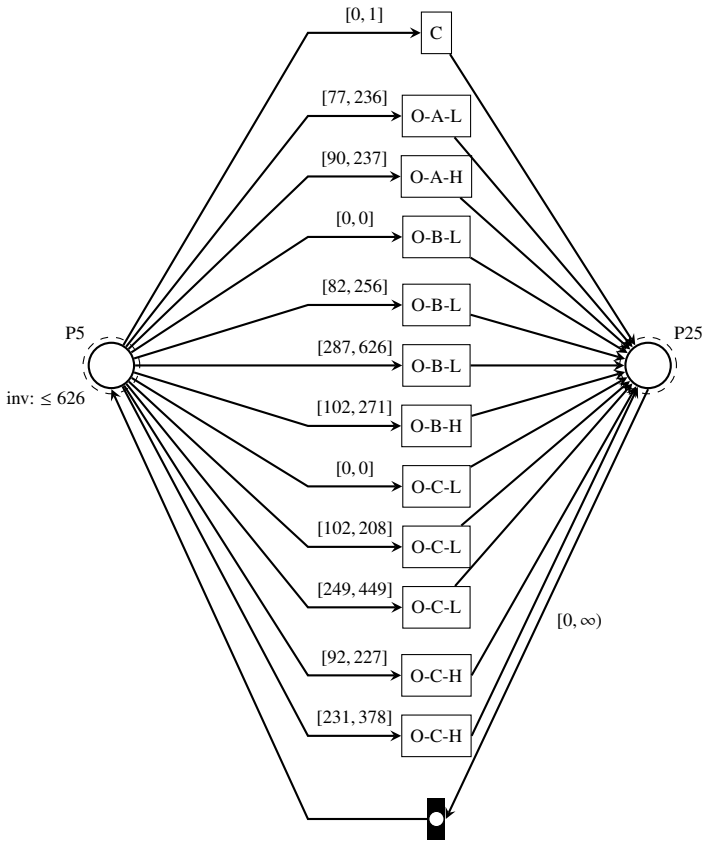


Figure 22: *Component 2 of workflow for w1 using the L2 log*

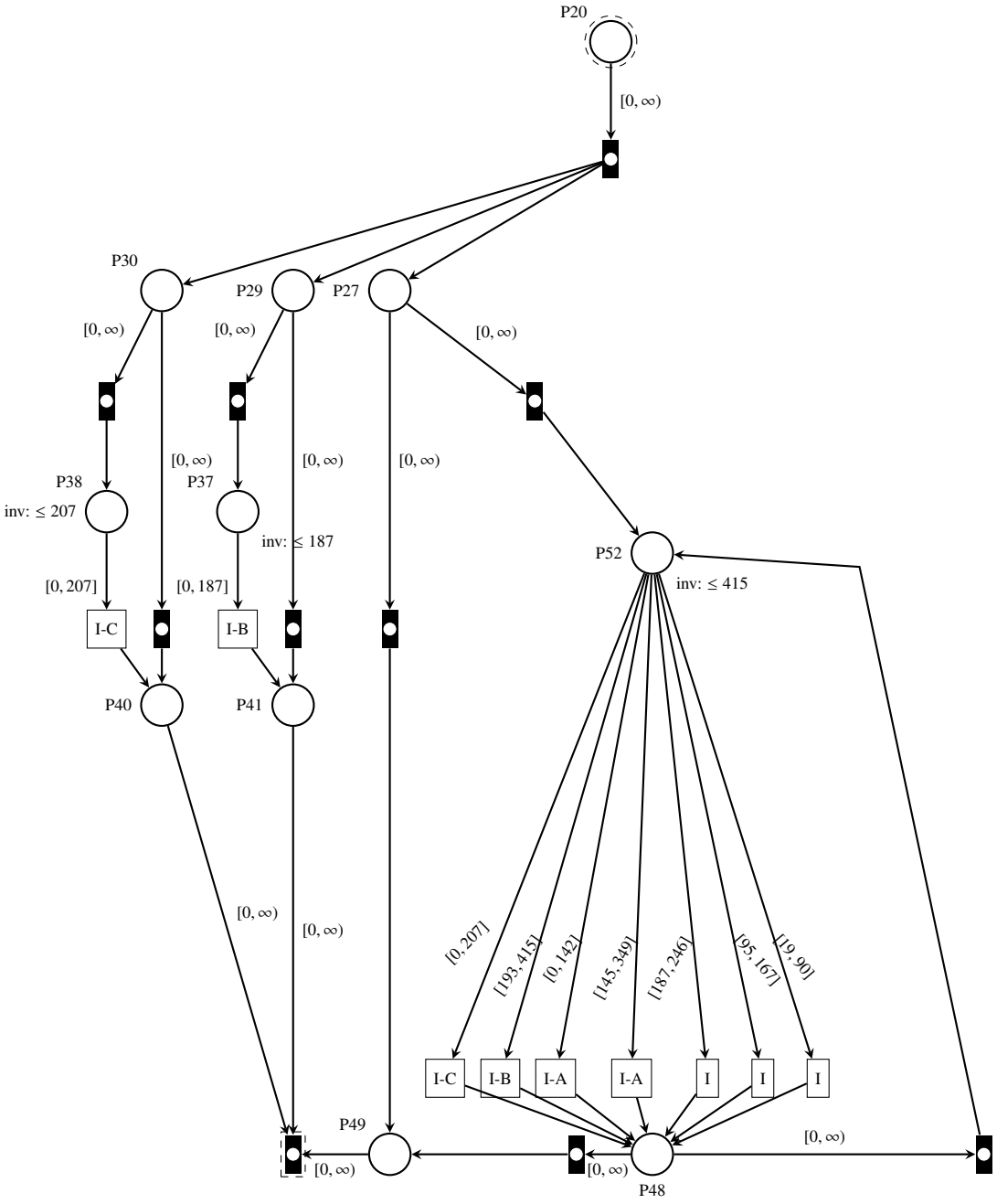


Figure 23: *Component 3 of workflow for w1 using the L2 log*

The next model we present, is generated on L2 for worker w2, where the model has been split into two components shown in Figures 24 and 25. Compared to the previous model, we see a more restricted workflow, where some actions must occur. E.G, atleast one O or C action must be executed, whereas there still is no guarantee for an I action to be executed. From comparing this model with the previous model for w1, we also see that there are fewer transitions in the new model. We believe this reduction of transitions is due to fewer orders in w2's log, and we deduce that this is one reason for the more precise model.

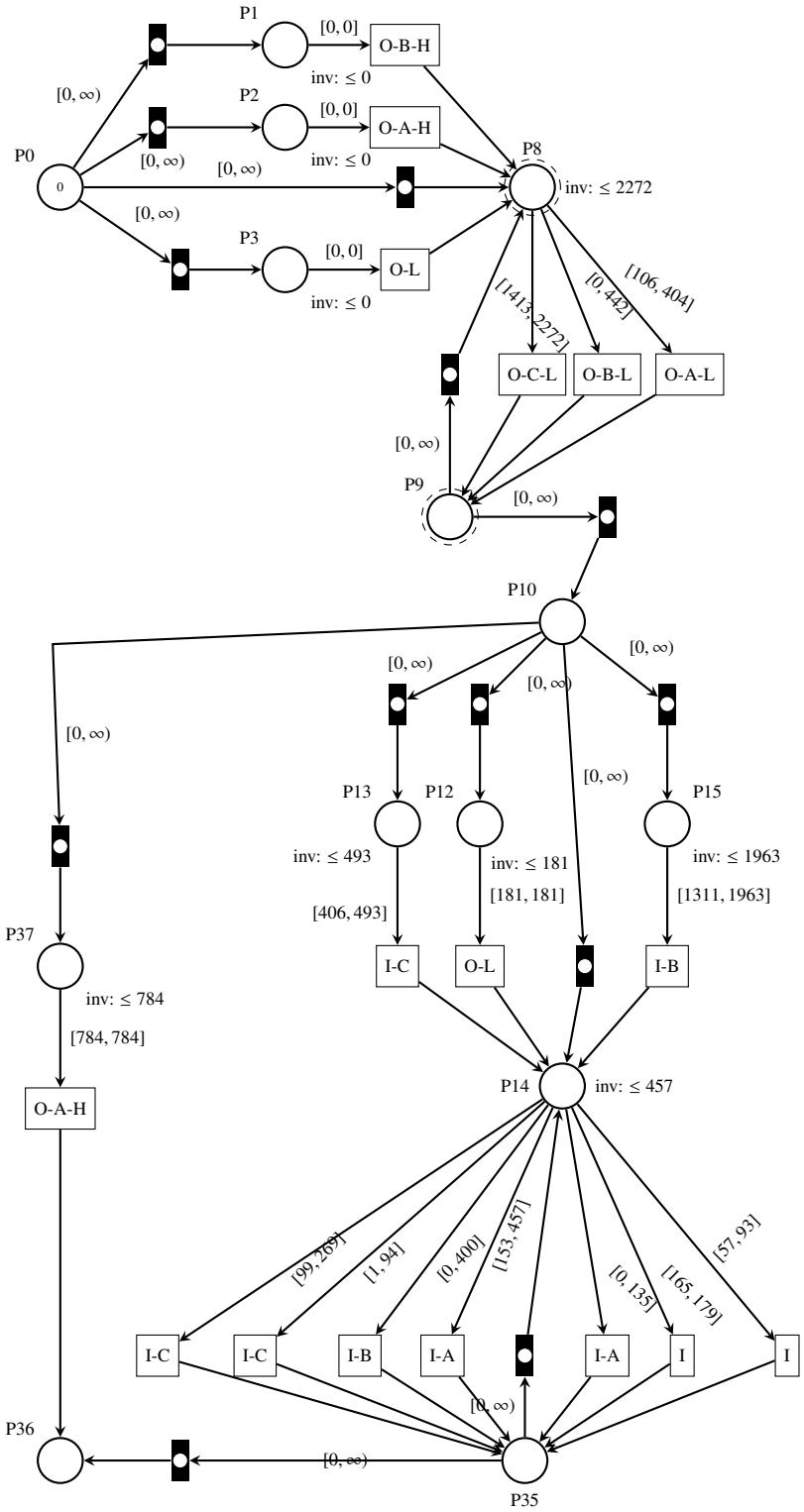


Figure 24: *Component 1 of workflow for w2 using the L2 log*

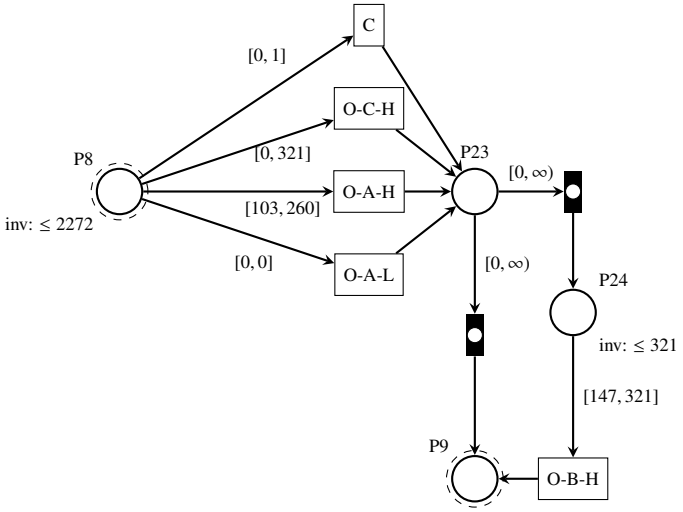


Figure 25: *Component 2 of workflow for w2 using the L2 log*

The final model generated for the L2 log is shown in Figure 26, which consists of the workflow for w3. In this workflow we see an increase in the number of transitions, when compared to the model for w2, which correlates with the increase in orders. At the same time, the model is not as general as the model for w1, e.g. as it is not possible to have a trace without atleast one C or O action.

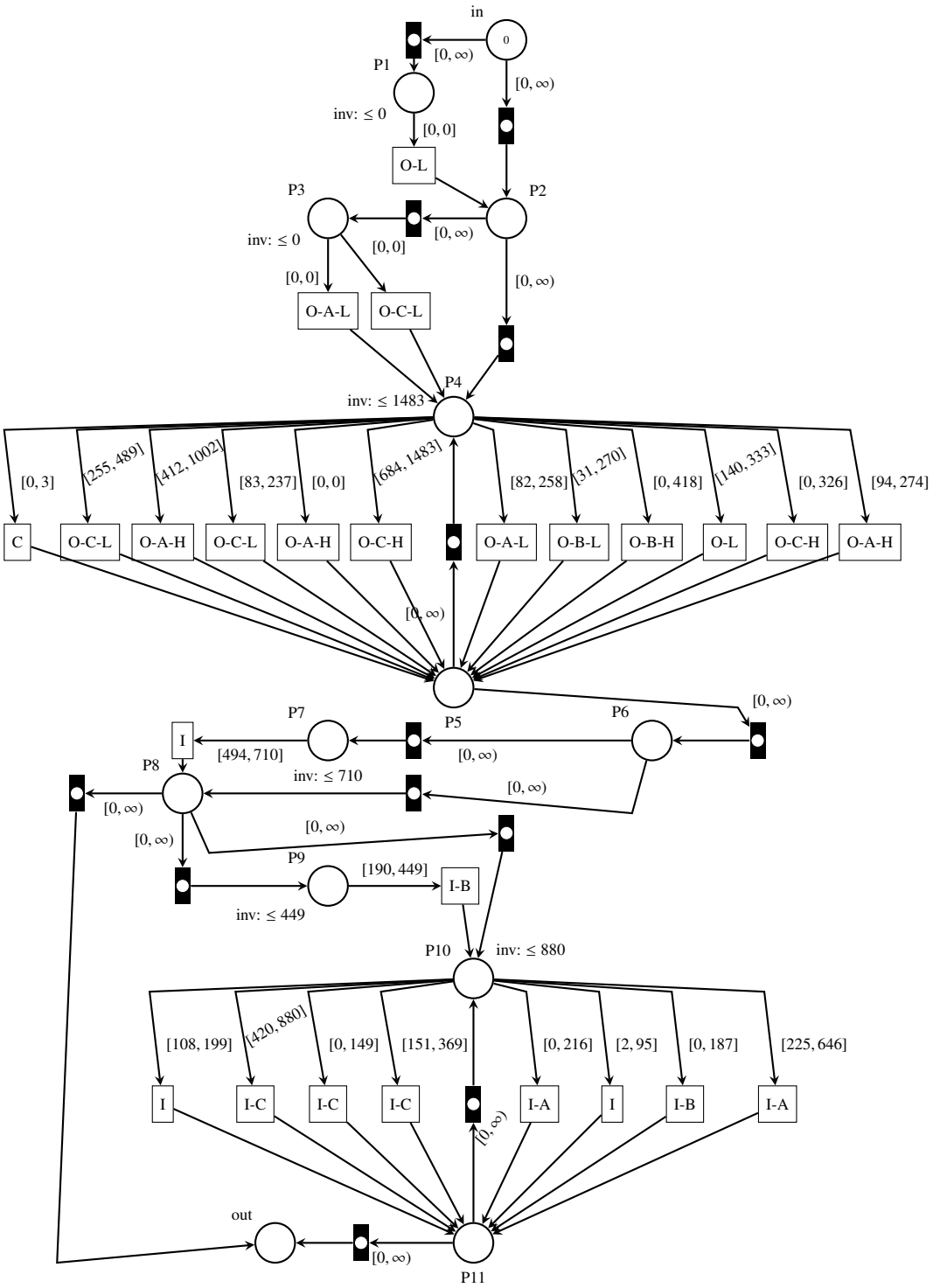


Figure 26: *Workflow for w3 using the L2 log*

13.5.2 Workflow models for workers on log L3

In the following we presents models for workers, which have been generated using the L3 log, i.e. there now is the possibility of performing the actions related to our swap implementation. As explained, we use ProM to generate Petri nets from the logs. When we generated the models on log L3 for workers w2 and w3, we got the models shown in Figures 27 and 28. These models have a large degree of generality, where there are no restrictions to the sequence of firing different transitions. We tried adjusting the parameters for generating the Petri nets and for the net for w2 we found that when adjusting the noise level to 50 %, a workflow began to form, but the amount of transitions was very restricted. For the Petri net for worker w3, we found that even with a noise level on 90 % the model still had the form as presented in Figure 28

Because of the high level of generality in the two models, we chose to disregard the models in the following work, and we will only consider the generated model for w1.

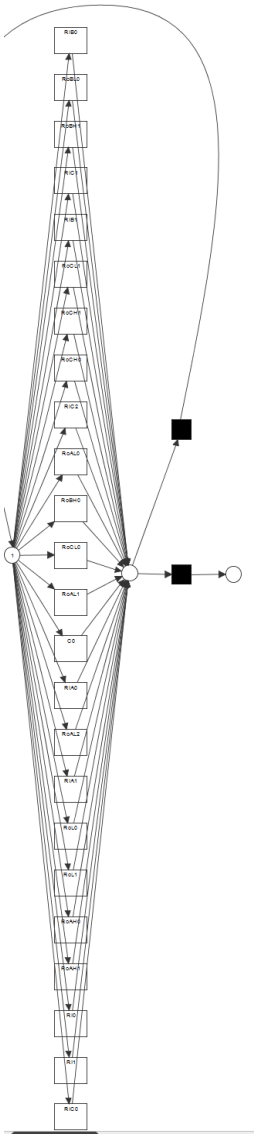


Figure 27: *flower model for w2 on L3*

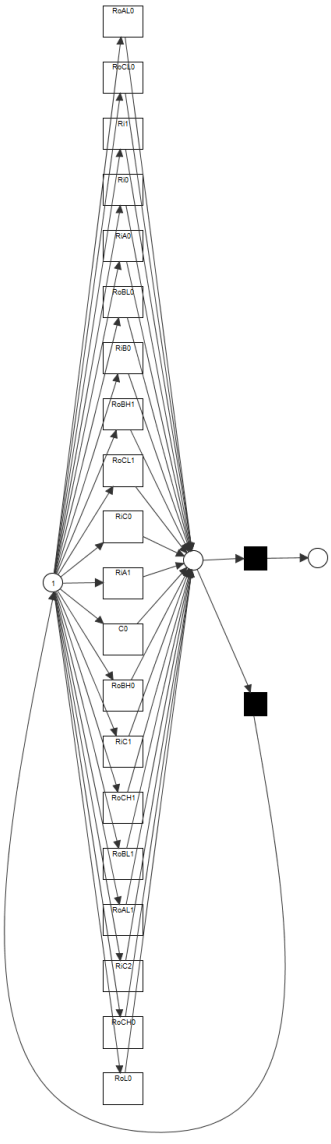


Figure 28: *flower model for w3 on L3*

The model that was generated for w1 is split into three components, shown in Figures 29 through 31. If we compare this model with w1's model over L2, then we see that the new model is still general, as it is still possible to traverse the entire net only firing τ-transitions. and we see a large amount of different transitions. However, the major difference between the two models is shown in Figure 31, as the new model now includes transitions related to performing swaps, i.e. we see that the worker has used our implementation. Furthermore we can observe that all swap related transitions may be fired independently of the other work performed by the worker, i.e. our notion, presented in Section 13.3, of a worker being able to perform swaps while he is not performing other tasks, holds. Also the labels for the swap related events ahve changes slightly, now we have Begin and End which signifies the first and last events related to performing swaps. And we include Tmp, Loc1 and Loc2, which depicts how the worker accessed the locations, where Tmp is the temporary location, Loc1 is the first location the worker accesses, and Loc2 is the last location he accesses. Lastly we added the specific heights accessed.

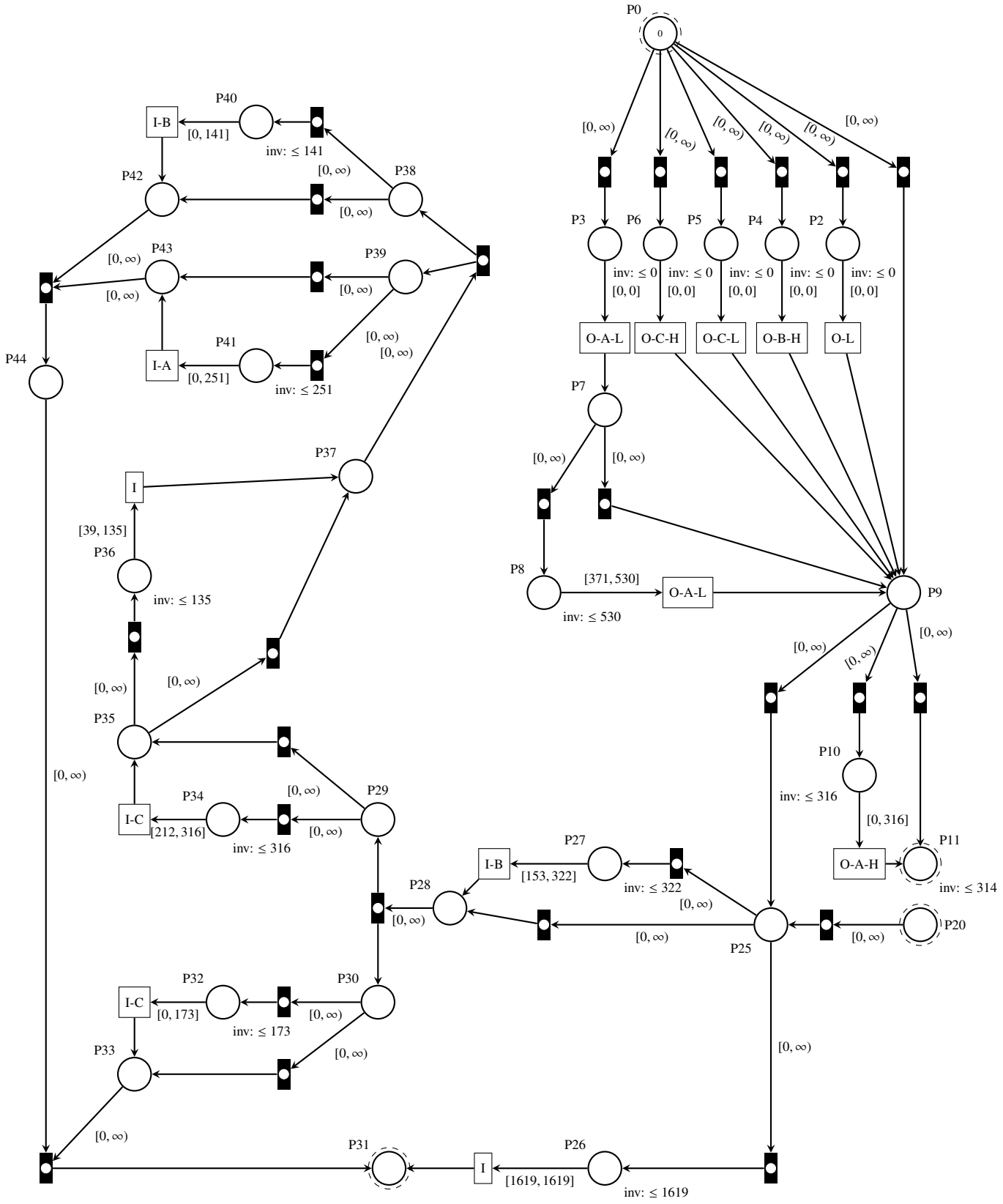


Figure 29: *Component 1 of workflow for w1 using the L3 log*

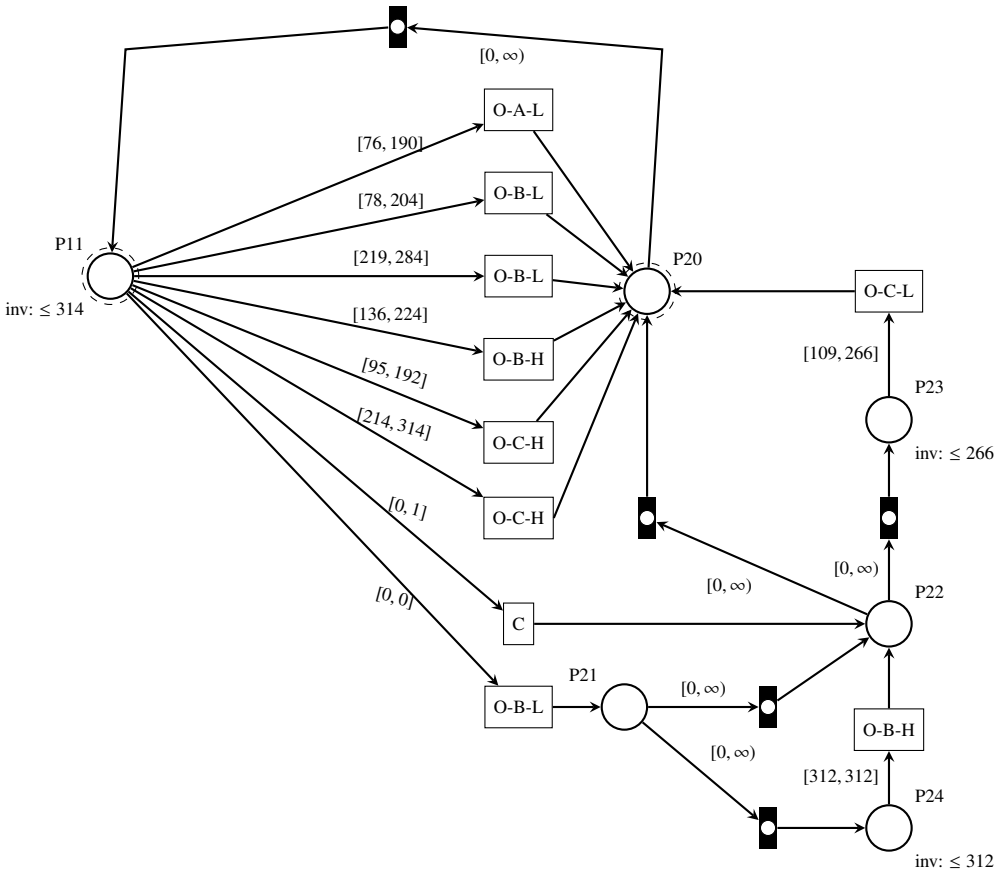


Figure 30: *Component 2 of workflow for w1 using the L3 log*

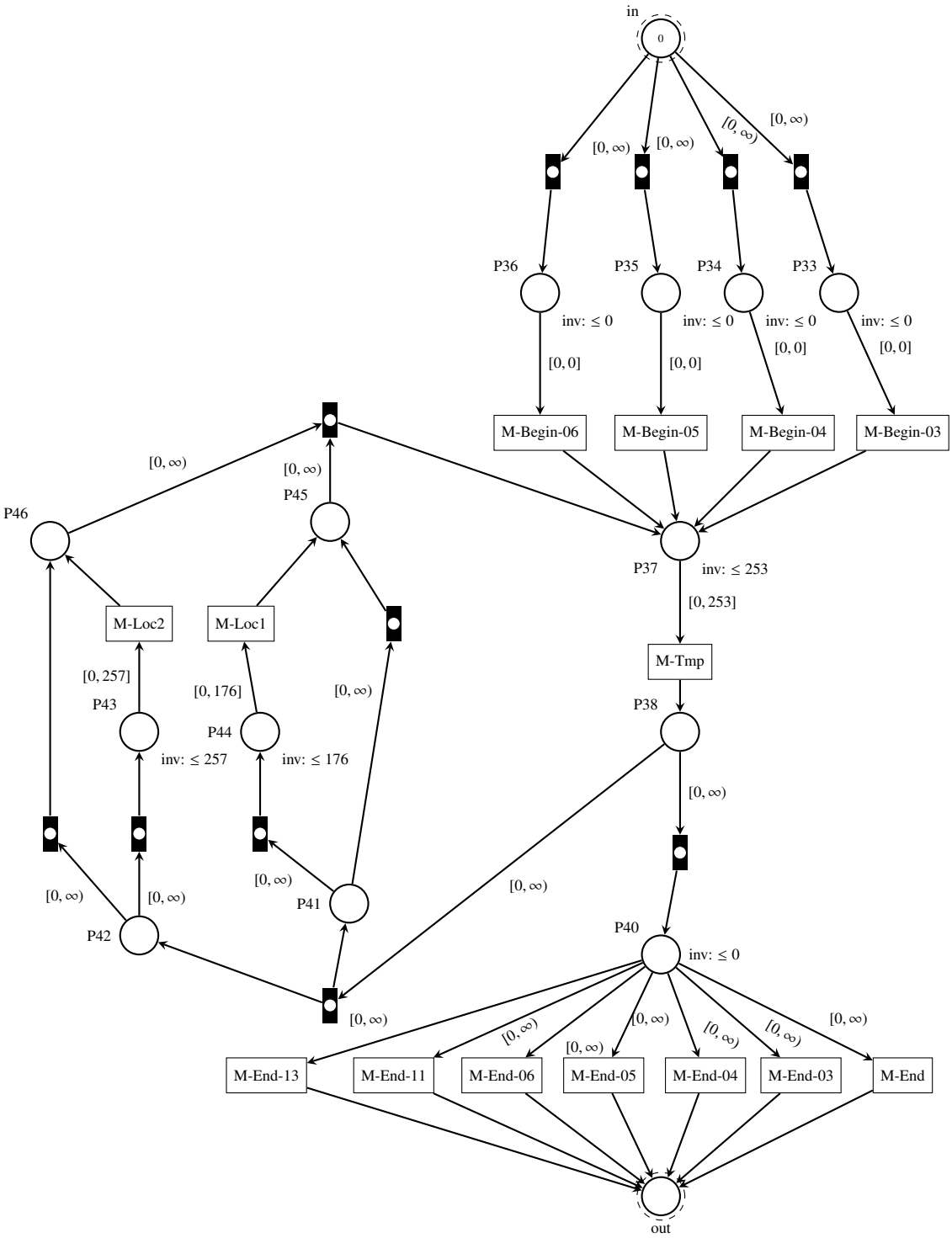


Figure 31: *Component 3 of workflow for w1 using the L3 log*

13.6. Evaluating Our Distance Algorithm

In the following experiment we shall evaluate our distance algorithm. In this experiment we shall use the test logs for worker w1, w2, and w3 from the L2 log. These test traces will be replayed, by the distance algorithm, on the models which we discovered using each worker's respective training data from the L2 log. By analysing the results of this experiment we expect to see that each worker's test log fits best with their own model, although this is not guaranteed. The reason for this may be solely due to a shift in worker behaviour, but contributing factors may also be found by assessing the worker's model. This is something we shall comment on when discussing the results of this experiment.

In Algorithm 5 we penalise for all enabled urgent transitions when performing a delay. This is something which we decided not to do in this experiment in order to isolate the time performance of each worker. By not penalising for τ-transitions, which in the models are used for routing purposes, we get a distance which indicates how well a worker performed based solely on their time performance with regards to a given model.

		L2 Models		
		w1	w2	w3
L2 Logs	w1	120	872	833
	w2	109	1479	1046
	w3	337	1501	571

Table 14: *The average distance resulting from replaying test traces for each worker on the worker models where model w1 = Figures 21 through 23, model w2 = Figures 24 and 25, and model w3 = Figure 26*

In Table 14 we show the average distance between a worker's test trace and each of the models which was discovered from the L2 log. In Table 14 the w1 is the model found in Figure 26, w2 is the model found in Figures 24 and 25, and the w3 model can be found in Figures 21 through 23. We show the average distance for the traces in the logs since there is a discrepancy between the number of logs handled by each worker in the period of the L2 log.

The first observation to make regards the w1 model (w1 column). Of the three workers it is not w1 that fits this model the best but w2. In fact, this model is the one that all the test logs fits with the best i.e. for all the worker logs it is the w1 model that has the lowest average distance. The cause of this may be found in the generality of the w1 model. In the w1 model it is possible to reach any non-τ-transition by only firing τ-transitions. This is not possible in the w2 model nor is it possible in the w3 model. Consequently, any trace from any of the workers can be replayed while only penalising on any cheat delays as long as the trace does not contain transition labels not present in the w1 model. This notion is cooperated by inspecting the ordered list of distances for each of the worker's test logs in relation to the three models as the w1 model consequently is the one with the most traces only penalised for cheat delaying.

As mentioned previously we expect each worker's test log to fit their own model the best i.e. they have the lowest distance to their own model. So far we have observed that the test log of w1 did in fact not fit best on the w1 model as the test log of w2 has a lower distance. This, however, does not disprove the correctness of our distance algorithm since; if we regard the test log of w1 in relation to each of the three models we see that it indeed fits best to the w1 model meeting our expectation.

In order to identify which test log model w1 fit best with we looked from the perspective of the test log (row perspective). If we look from the test log perspective to identify the model best fitting test log w3 we find that the w1 model has the lowest distance. This, however, is misleading and occurs since the w1 model was the most general model of the three models and therefore may explain the test log of w3 well—and in our case even better than the w3 model. If we adopt a model perspective (column perspective) and focus on the w3 model column we correctly observe that the test trace of w3 has the lowest value. This dichotomy between the test log and model perspective represents a difficult choice. It may be the case that if we had

another model with high generality we could not correctly pair the w1 test trace with the w1 model.

Both perspectives falls short of pairing the w2 test log and w2 model. This might be due to the low number of traces in the log for the w2 worker. In the L2 log we have observed the following amount of traces for each of the workers: w1 968 traces, w2 118 traces, and w3 817 traces. When we split the w2 worker's data into training and test logs we have a split of 95 training traces and 23 test traces, where as w1 has 775 training traces and 196 test traces and w3 654 training traces and 163 test traces. As a consequence of the low data amount for w2 the model becomes more restricted to the behaviour observed in the training data. We see this clearly as both the test log for w1 and w3 fits better on the w2 model than w2's test trace indicating that a shift in behaviour is the most likely cause. In order to determine which perspective facilitates the pairing of test logs with their respective worker models is something that must be examined further.

14. Conclusion

In Section 3 we outlined the contribution of our thesis which is divided into two separate, but closely related, functions of operating a warehouse. The first function which we mainly focus on in Part I, regards the assignment of items to locations in a warehouse, which is generally referred to as the storage location assignment problem. The second function of operating a warehouse, which we contribute to, regards the analysis of workflows where we employ the theory of process mining. In Part II we focus on this second function of the warehouse and use process mining in order to evaluate our contribution to solving the storage assignment problem.

Our initial efforts in Part I focused on understanding the problem itself and on defining the context in which the problem occurs i.e. what actually happens in a warehouse. A vital component in gaining insight into the storage problem and its context came from our stakeholders, presented in Section 5. These stakeholders were the warehouse Av Form and the company Logimatic—the developers of the warehouse management system LOGIA. Through these stakeholders we were able to define a concrete area suitable for optimisation, namely the allocation of carriers to storage locations. The concrete problem defined in collaboration with Av Form, was to research the possibility of rearranging locations referred to as reserve locations, in order to reduce the total time spend on replenishment—a task related to order picking.

In Section 6 we described the theoretical basis for how workers at Av Form can perform actions that affect the configuration of a warehouse. Afterwards in Section 7 we devised a novel algorithm for suggesting pairs of locations to be swapped such that the overall time used on replenishment is reduced. With the algorithm well defined, we presented some time estimates needed for the initial implementation of it in to LOGIA. These time estimates and the implementation was described in Sections 8 and 9 respectively. Our LOGIA implementation was installed at and used by Av Form for a short duration, which generated event logs that we used for Part II of our thesis.

Then for Part II we first presented the theory of Petri nets and Extended Timed-Arc Workflow nets (ETAWFNs) in Section 10, such that we in the following Section 11 could present an issue uncovered during our previous work when applying process mining to discrete-time event logs (DTELs). The issue arose during our previous work mainly due to the use of two different process mining tools during the discovery process which used different modelling languages. The issue, elaborated on in Section 11.1, resulted in a lack of precision in the discovered ETAWFN with regards to time discrimination. In order to solve this issue we proposed two methods: Clustering of delays and concatenating action types. The concatenating method was rather simple, but for the clustering method we devised our own algorithm in Section 11.2 which used the k-means algorithm and box plot theory to decide which time values to use in the discovered ETAWFN. Then in order to examine the effectiveness of the two proposed methods we presented a previous contribution of ours in Section 12 pertaining to the field of process mining.

The final section of Part II presented four different experiments evaluating our contributions. The first experiment in Section 13.3 examined the impact of our swap suggestions using logs from our case study. We

found that under the assumption that swaps are performed in a worker's idle time it will result in a reduction in the time spend on retrieving items from replenish locations, in the warehouse. The second experiment used our distance algorithm to evaluate our proposed methods to the time dilemma. We found that the clustering method and the combination of concatenating and clustering significantly outperformed both our old method, and only applying concatenating. Of the two well performing methods we chose to use clustering since it results in models with a higher degree of simplicity, although it is not as precise as using the combined method. Then for the third experiment, we examined the workflows of Av Form where we looked at three different workers. We found that if locations were swapped it was done separately from the main workflow, i.e. adding to our assumption that swaps could be performed during a workers idle time, hence not adding to the swapcost. Secondly, we found the workflows do not strictly follow what is expected from Logimatic, as they expected replenish related events to happen concurrently, whereas we observed that this was not the case in either of the workflows. However, we believe this is most likely due to workers overtaking orders from other workers, which results in LOGIA loosing track of the workflow. For our last experiment, Section 13.6, we evaluated our distance algorithm on the models discovered in experiment three. We found that we in two out of three cases could correctly pair a worker with their own workflow. The case where it was not possible, we believe is a result of a small dataset for the specific worker, or that it would require examination from another angle in order for our evaluation of the distance algorithm to be conclusive with regards to the specific worker.

In conclusion for the first part of our thesis, we have shown through theory and a solution applied at a warehouse, that we are able to rearrange reserve locations, which results in a reduction to the total time needed for replenishment, and thereby satisfying the problem stated in collaboration with Av Form.

For the second part, we conclude that we have been able to address the discovered problem related to time discrimination, and through experiment two, we showed how two of the suggested solutions provided results, which showed a finer granularity with regards to time discrimination. Finally, in the fourth experiment, we were able to show how our use of the distance measure may be used to analyse and compare ETAWFNs and DTELs, though seemingly with the restriction that datasets must be of some minimum size.

15. Future Work

In Section 7 we presented our swap suggestions algorithm for which we have identified aspects that could be improved. The most significant of which is that the reduction in cost it computes has been shown in relation to our case study in Section 13.3 to not be realised within a three months period, if the cost of swapping two locations is also considered. We contribute this to the low access frequency for the swapped locations combined with the low benefit from moving carriers to lower locations. In order to improve the swap suggestion algorithm we propose to extend it such that it considers other aspects of the replenishment task. In its current incarnation it only considers the time it takes the worker to bring a carrier down to the ground. One aspect to incorporate into the algorithm would be the time it takes the worker, once a carrier has been brought down, to move an item to its forward location. The time it takes to move an item to its forward location is, however, not solely dependent on the most direct path to its forward location. The routing pattern which the warehouse employs affects the time as well. In the case of Av Form with their s-pattern it would not be sufficient to just move items close to their forward location, but close to it and before it. This way a worker may drop off an item when passing by its forward location avoiding any backtracking to forward locations. There are also other patterns than the s-pattern; hence, another point of improvement for the swap suggestion is to generalise it to more than the s-pattern.

The swap suggestion algorithm may also be extended such that it considers which items are picked together and attempt to store these close to one another. Koster et al. [3] refers to this notion as family grouping where the relation between items is part of the storage assignment process. A simple metric to use in this problem is the frequency at which items appear together. Given such a metric the authors mention two

types of family grouping: Complementary-based method, and contact-based. These two methods are evident contenders in our future work, but the research into the existence and effectiveness of other methods is also necessary.

In Section 13.4 we examined four preprocessing methods with the aim of improving the discovery process. We found that the clustering and the combination of concatenating followed by clustering introduced significant improvements where the combined method outperformed solely applying clustering. The precision gained under the combined method comes at a cost of model complexity. This fact coupled with us manually constructing ETAWFNs from Petri nets motivated our decision to continue with the clustering method over the combined method. To use the combined method in our future work it will require an automated process to construct ETAWFNs from Petri nets.

Another more theoretical improvement to the evaluation of the preprocessing methods is to evaluate them with regards to what Aalst [9] refers to as quality attributes. The author presents four aspects to consider when assessing the quality of a model: fitness, simplicity, precision, and generalisation. In our thesis we briefly mentioned simplicity when discussing the time dilemma. Furthermore, we have already assessed the preprocessing methods based on fitness since our distance algorithm quantifies this quality attribute for a given discrete-time event log and timed-arc workflow net. However, since the four quality attributes affect one another, e.g. a more general model is less precise, it would also be beneficial to include the two yet unexplored quality attributes of generalisation and precision. Finally, The author defines the quality attributes for untimed models; hence, it would also be necessary to formally define them in relation to time.

References

[1] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. "An overview of workflow management: From process modeling to workflow automation infrastructure". In: *Distributed and parallel Databases* 3.2 (1995), pp. 119–153.

[2] Ewa Deelman et al. "Workflows and e-Science: An overview of workflow system features and capabilities". In: *Future generation computer systems* 25.5 (2009), pp. 528–540.

[3] René de Koster, Tho Le-Duc, and Kees Jan Roodbergen. "Design and control of warehouse order picking: A literature review". In: *European Journal of Operational Research* 182.2 (2007), pp. 481–501. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2006.07.009. URL: http://www.sciencedirect.com/science/article/pii/S0377221706006473.

[4] Yipeng Zhang. "Correlated Storage Assignment Strategy to reduce Travel Distance in Order Picking". In: *IFAC-PapersOnLine* 49.2 (2016), pp. 30–35.

[5] Monika Kofler et al. "Rassigning storage locations in a warehouse to optimize the order picking process". In: *Proceedings of the 22th European Modeling and Simulation Symposium EMSS*. 2010, pp. 77–82.

[6] Nikolaos Tsamis et al. "Adaptive storage location assignment for warehouses using intelligent products". In: *Service Orientation in Holonic and Multi-agent Manufacturing*. Springer, 2015, pp. 271–279.

[7] Martin Hilbert and Priscila López. "The worlds technological capacity to store, communicate, and compute information". In: *science* 332.6025 (2011), pp. 60–65.

[8] Foster Provost and Tom Fawcett. "Data science and its relationship to big data and data-driven decision making". In: *Big Data* 1.1 (2013), pp. 51–59.

[9] Wil MP van der Aalst. *Process mining: data science in action*. 2nd ed. Springer, 2016.

[10] W.M.P. van der Aalst et al. "Business process mining: An industrial application". In: *Information Systems* 32.5 (2007), pp. 713–732. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2006.05.003. URL: http://www.sciencedirect.com/science/article/pii/S0306437906000305.

[11] Gideon Jonas Baumann Blegmand and Christian Stephansen. "Conformance Checking Extended Timed-Arc Workflow Nets". In: (2018).

[12] JEROEN P. Van den Berg. "A literature survey on planning and control of warehousing systems". In: *IIE Transactions* 31.8 (Aug. 1999), pp. 751–762. DOI: 10.1023/A:1007606228790. URL: https://doi.org/10.1023/A:1007606228790.

[13] O. Cruz-Dominguez and R. Santos-Mayorga. "Artificial intelligence applied to assigned merchandise location in retail sales systems". en. In: *South African Journal of Industrial Engineering* 27 (May 2016), pp. 112–124. ISSN: 2224-7890. URL: http://www.scielo.org.za/scielo.php?script=sci_arttext&pid=S2224-78902016000100010&nrm=iso.

[14] Peng Yang et al. "Variable neighborhood search heuristic for storage location assignment and storage/retrieval scheduling under shared storage in multi-shuttle automated storage/retrieval systems". In: *Transportation Research Part E: Logistics and Transportation Review* 79 (2015), pp. 164–177.

[15] Jinxiang Gu, Marc Goetschalckx, and Leon F McGinnis. "Research on warehouse operation: A comprehensive review". In: *European journal of operational research* 177.1 (2007), pp. 1–21.

[16] Sander JJ Leemans, Dirk Fahland, and Wil MP van der Aalst. "Discovering block-structured process models from event logs containing infrequent behaviour". In: *International Conference on Business Process Management*. Springer. 2013, pp. 66–78.

[17] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. "Discovering Block-Structured Process Models from Event Logs - A Constructive Approach". In: *Application and Theory of Petri Nets and Concurrency*. Ed. by José-Manuel Colom and Jörg Desel. Springer Berlin Heidelberg, 2013, pp. 311–329.

[18] The Process Mining Group at Eindhoven Technical University. *ProM*. URL: http://www.promtools.org/doku.php (visited on 12/17/2017).

[19] Av Form. *Av Form*. 2017. URL: http://www.avformshop.dk/ (visited on 10/31/2017).

[20] Logimatic. *LOGIA*. 2017. URL: http://www.logiawms.com/ (visited on 10/31/2017).

[21] Logimatic. *Logimaitc*. 2017. URL: http://www.logimatic.com/ (visited on 01/03/2018).

[22] AJMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. "Process mining with the heuristics miner-algorithm". In: *Technische Universiteit Eindhoven, Tech. Rep. WP* 166 (2006), pp. 1–34.

[23] Shane Frederick, George Loewenstein, and Ted O'donoghue. "Time discounting and time preference: A critical review". In: *Journal of economic literature* 40.2 (2002), pp. 351–401.

[24] Erik Meijering. "A chronology of interpolation: From ancient astronomy to modern signal and image processing". In: *Proceedings of the IEEE* 90.3 (2002), pp. 319–342.

[25] Microsoft. *Introduction to WPF*. URL: https://msdn.microsoft.com/da-dk/library/aa970268(v=vs.100).aspx (visited on 05/12/2018).

[26] Microsoft. *The MVVM Pattern*. 2012. URL: https://msdn.microsoft.com/en-us/library/hh848246.aspx (visited on 05/12/2018).

[27] Michael J McLaughlin, John Harper, and Joseph McLaughlin. *Oracle Database 12c PL/SQL Programming*. McGraw-Hill Education, 2014.

[28] Oracle. *Oracle Database 18c PL/SQL*. URL: http://www.oracle.com/technetwork/database/features/plsql/index.html#close (visited on 05/15/2018).

[29] *Ask TOM "Context Switch"*. URL: https://asktom.oracle.com/pls/apex/f?p=100:11:0::::p11_question_id:6012271510360 2 (visited on 05/15/2018).

[30] *Database PL/SQL User's Guide and Reference*. URL: https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/cursor_declaration.htm (visited on 05/18/2018).

[31] José Antonio Mateo, Jir Srba, and Mathias Grund Sørensen. In: *Fundamenta Informaticae* 140.1 (2015), pp. 89–121.

[32] Wil MP Van Der Aalst et al. "Soundness of workflow nets: classification, decidability, and analysis". In: *Formal Aspects of Computing* 23.3 (2011), pp. 333–363.

[33] Wil MP Van der Aalst. "The application of Petri nets to workflow management". In: *Journal of circuits, systems, and computers* 8.01 (1998), pp. 21–66.

[34] fluxicon. *Disco*. URL: https://fluxicon.com/disco/ (visited on 12/17/2017).

[35] Depatment of Computer Science at Aalborg University. *TAPAAL*. URL: http://www.tapaal.net/ (visited on 12/18/2017).

[36] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. "Data clustering: a review". In: *ACM computing surveys (CSUR)* 31.3 (1999), pp. 264–323.

[37] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 29. wh freeman New York, 2002.

[38] Brian Hayes. "Computing science: The easiest hard problem". In: *American Scientist* 90.2 (2002), pp. 113–117.

[39] Peter C Cheeseman, Bob Kanefsky, and William M Taylor. "Where the really hard problems are." In: *IJCAI*. Vol. 91. 1991, pp. 331–340.

[40] Stuart Lloyd. "Least squares quantization in PCM". In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[41] David Arthur and Sergei Vassilvitskii. "How slow is the k-means method?" In: *Proceedings of the twenty-second annual symposium on Computational geometry*. ACM. 2006, pp. 144–153.

[42] Sariel Har-Peled and Bardia Sadri. "How fast is the k-means method?" In: *Algorithmica* 41.3 (2005), pp. 185–202.

[43] Kristin Potter et al. "Methods for presenting statistical information: The box plot". In: *Visualization of large and unstructured data sets* 4 (2006), pp. 97–106.

[44] Jiri Srba and Peter Gjøl Jensen. *VerifyDTAPN*. Revision: 378, Branch: https://code.launchpad.net/ verifydtapn-contributers/verifydtapn/partial-order. 2012. URL: https://launchpad.net/verifydtapn.

[45] David L Poole and Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.

[46] Robert E. Sanders. "The Pareto Principle: Its Use And Abuse". English. In: *The Journal of Services Marketing* 1.2 (1987). URL: https://search.proquest.com/docview/212681646?accountid=8144.

Appendices

A. Concatenated Discrete-Time Event Log

In Table 15 we have concatenated the event type of each event in a given trace with the preceding event's type.

Type	Location	Item	Quantity	Worker	Order number	Delay
C	859-05-03-B	47	33	avr	1	0
CO	859-05-03-B	47	-10	avr	1	11
C	444-01-03-B	15	4	bob	2	19
CO	444-01-03-B	15	-4	bob	2	33
M	858-03-11-A	74	-32	avr	3	71
MM	859-01-11-B	74	32	avr	3	71
MM	858-03-11-A	97	-15	avr	3	71
MM	859-01-11-B	97	15	avr	3	71
OO	853-01-01-C	46	-109	bob	2	118
MO	227-05-03-B	8	-23	avr	3	157
M	308-04-03-C	16	-87	cre	4	192
MM	441-18-03-B	16	87	cre	4	192
MM	308-04-03-C	69	-94	cre	4	192
MM	441-18-03-B	69	94	cre	4	192
OO	441-12-04-A	11	-98	bob	2	201
MO	441-18-03-B	16	-45	cre	4	270
OO	439-05-03-A	17	-31	cre	4	368

Table 15: *Table 3 where events have been concatenated*