PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

DERIVING CONFIGURABLE PROCESS MODELS USING PROCESS MINING

MAURICIO JAVIER ARRIAGADA BENÍTEZ

Thesis submitted to the Office of Research and Graduate Studies in partial fulfillment of the requirements for the Degree of Doctor in Engineering Sciences

Advisors:
MARCOS SEPÚLVEDA FERNÁNDEZ
JORGE MUÑOZ GAMA

Santiago de Chile, April 2019

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

DERIVING CONFIGURABLE PROCESS MODELS USING PROCESS MINING

MAURICIO JAVIER ARRIAGADA BENÍTEZ

Thesis submitted to the Office of Research and Graduate Studies in partial fulfillment of the requirements for the Degree of Doctor in Engineering Sciences

Santiago de Chile, April 2019

To my family

ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for giving me the strength, knowledge, opportunity and perseverance to undertake this research study and complete it satisfactorily. I never doubted about where he is leading me.

I would like to express my special thanks of gratitude to my advisors Prof. Sepúlveda and Prof. Muñoz who always supported me during the research work. Both showed and guided me on how to improve my research.

To my family who were at all times with me in each of the stages of these years of studies. I will be eternally grateful for their unconditional support, especially to my mom who taught me how to face new challenges, to my sister who always encouraged me to achieve new challenges as well, and to my dad who taught me to be persistent at work. To my wife for accompanying and supporting me in this important part of my life with all her love and knowledge. All of you have all made a tremendous contribution in helping me reach this stage in my life.

This thesis was supported by Comisión Nacional de Investigación y Tecnología CONICYT - Ministry of Educacation, Chile, Ph.D. Student Fellowships Folio 21120850. I would also like to express my sincere gratefulness to CONICYT.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

RESUMEN

Utilizado con frecuencia en grandes organizaciones con sucursales en diferentes ubicaciones, un modelo de proceso configurable reune las características más comunes que se comparten entre diferentes sucursales. Este modelo de proceso configurable puede configurarse para derivar un modelo de proceso específico para cada sucursal. El proceso de configuración generalmente se realiza de forma manual, lo cual es un reto por dos razones. Por un lado, cuando el número de puntos de configuración aumenta en el modelo de proceso configurable, el tamaño del espacio de búsqueda aumenta exponencialmente. Por otro lado, la persona que realiza la configuración puede carecer de una perspectiva holística para tomar la decisión correcta para todos los nodos configurables. Hoy en día, en muchos escenarios de negocios, los sistemas de información que apoyan la ejecución de procesos de negocios crean registros de eventos, que contienen datos que reflejan cómo se están realizando los procesos. En esta tesis, proponemos tres estrategias que utilizan estos datos de eventos para derivar automáticamente un modelo de proceso a partir de un modelo de proceso configurable, tal que el modelo generado sea el que mejor representa las características del proceso en una sucursal específica. La primera estrategia se basa en una búsqueda exhaustiva, la segunda se basa en un enfoque evolutivo genético y la tercera se basa en una heurística codiciosa. Hemos implementado estas tres estrategias diferentes en ProM como parte de nuestra propuesta para derivar un modelo de proceso. Hemos probado estas estrategias utilizando registros de eventos realistas que representan el comportamiento de diferentes variantes de proceso, tal como se registran en un sistema de educación superior, y también utilizamos un caso real de municipalidades en Holanda.

Palabras Claves: minería de procesos, proceso de negocio, árbol de proceso, árbol de proceso configurable, modelo de proceso configurable, registro de eventos.

ABSTRACT

Frequently used in large organizations with branches across different locations, a configurable process model unifies the commonalities shared by all branches. This configurable process model can later be configured to derive a specific process model for each branch. Configuration is usually done manually, which is challenging for two reasons. On the one hand, when the number of configurable nodes in the configurable process model grows, the size of the search space increases exponentially. On the other hand, the person performing the configuration may lack the holistic perspective to make the right choice for all configurable nodes. Nowadays, in many business scenarios, information systems that support the execution of business processes create event logs, data reflecting how processes are being performed. In this thesis, we propose three strategies that use this event data to automatically derive a process model from a configurable process model that better represent the characteristics of the process in a specific branch. The first strategy is based on an exhaustive search, the second one is based on a genetic evolutionary approach, and the third one is based on a greedy heuristic. We have implemented these three different strategies in ProM, as part of our proposed framework to derive a process model. We have tested them using realistic event logs that represent the behavior of different process variants as recorded in a higher educational ERP system, and also using a real case scenario of Dutch municipalities.

Keywords: process mining, business processes, process tree, configurable process tree, configurable process model, event logs.

1. INTRODUCTION

Business process models and other discrete event systems are widely used for analysis, optimization, monitoring, and even auditing, as they describe the operations of an organization (van der Aalst, van Hee, van der Werf, & Verdonk, 2010). Often, variants of the same process occur in large organizations as a result of legal restrictions, cultural conditions, business strategies, and economical issues, among others. For example, banks commonly have branches in different locations where they use similar processes that might slightly differ in order to adapt to local conditions. Similarly, municipalities provide the same services, but the processes might differ significantly depending on the size of the population.

The challenge for these organizations is to balance standardization and a certain level of flexibility in their business processes. A process model that describes both the commonalities shared by all process variants and their differences is called a configurable process model. Extensions of process modeling languages have been developed in order to represent configurable process models, such as *C-YAWL* (Gottschalk, van der Aalst, Jansen-Vullers, & Rosa, 2008), *C-BPEL* (Ghedira & Mezni, 2006), *C-EPC* (Rosemann & van der Aalst, 2007), *C-PT* (Schunselaar, Leopold, Verbeek, van der Aalst, & Reijers, 2014), and *C-BPMN* (Sharma & Rao, 2014).

The importance of having a configurable process model to describe process variants has been a subject of study in the literature. For example, three of the twenty Business Process Management (BPM) use cases identified by van der Aalst (2012), involve configurable process model: *design configurable model (DesCM)*, *merge models into configurable model (MerCM)*, and *configure configurable model (ConCM)* (see Figure 1.1). In particular, the use case *ConCM* consists of deriving a process model from a configurable process model so as to represent a particular process variant. As shown in Figure 1.1[1], the original use cases a) manually design a configurable process model, b) merge collection of process models to generate a configurable process model, and c) configure a configurable

[1]This figure also shows our proposal d) as a new use case: to derive a process model based on a configurable process model and an event log.

process model to obtain a process model do not consider the usage of other sources of information beyond manual configuration.

However, in the last few years, a new discipline called process mining has emerged, which studies extracting and analyzing data recorded in information systems about processes' behavior (van der Aalst, 2016). More specifically, process discovery techniques aim at creating a process model based on the historical behavior recorded in an event log. Some process mining techniques have been applied to support the creation and derivation of configurable process models (Buijs, van Dongen, & van der Aalst, 2013), allowing to create a configurable process model based on several event logs (each containing the historical behavior of a particular business scenario) and to obtain a derived process model for each of the initial event logs. However, it does not allow to obtain a derived process model from the configurable process model for a different event log.

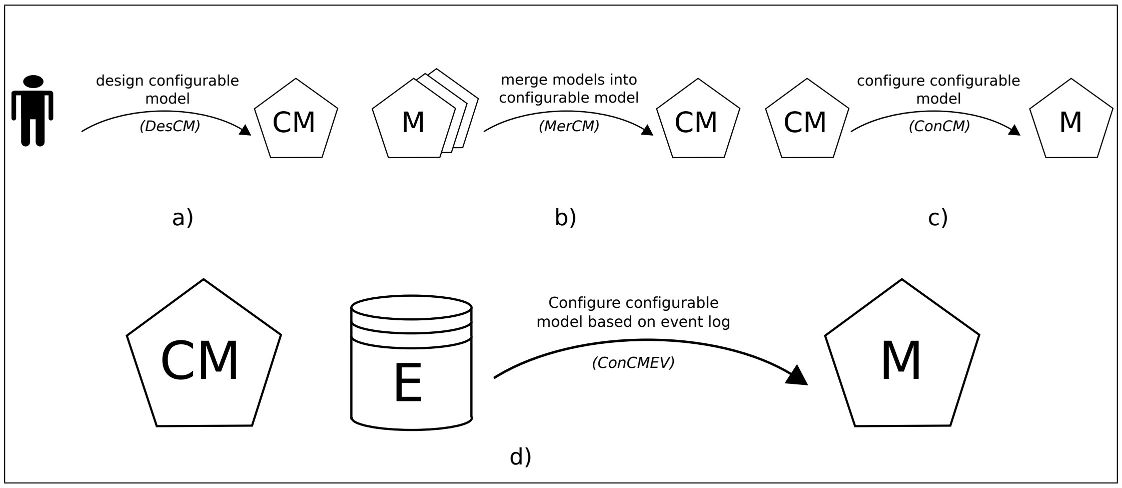

design configurable model
(DesCM)
CM
M
merge models into configurable model
(MerCM)
CM
CM
configure configurable model
(ConCM)
M
a)
b)
c)
CM
E
Configure configurable model based on event log
(ConCMEV)
M
d)

FIGURE 1.1. BPM use cases related to configurable process models.

While analyzing the literature about configurable process models, we have identified two main opportunities. As mentioned before, two separate approaches can be recognized: on the one hand, automated process discovery, and on the other hand, manual configuration of configurable process models. The first opportunity is to integrate these two approaches. At the same time, there is a growing availability of event data in organizations, which also promotes for having specialized, smart and efficient techniques to include historical data for configuring a configurable process model.

1.1. Research question

Based on these opportunities, we would like to address two challenges. The first and main challenge is to develop an automatic method to derive a process model from a configurable model that better represents the observed behavior of a process variant (e.g., the process execution in a target branch) as stored in a given event log, which can be applied on real-life data.

It is also important that configuration decisions can be defined in a local context. Hence, the second challenge is to create a simple representation for the different degrees of freedom we want to allow in the configurable nodes (Schunselaar, Verbeek, van der Aalst, & Reijers, 2012).

Therefore, the main research question in this thesis is the following:

How can a process model be derived automatically from a configured process model and an event log?

The advantage of automatically deriving a process model from a configurable process model that better represents the behavior observed in an event log (instead of discovering a process model only based on the behavior observed in the event log) is that it is useful when an organization has already defined a configurable process model in order to specify a certain degree of standardization and flexibility among the different branches where the process is being performed. If a new branch is incorporated, it would be desirable to obtain a derived process model that is as similar as possible to the current way of executing the process in such a branch, represented by an event log. While it is possible to use any process discovery technique based on the event log to obtain a process model that will describe how the process is currently executed in the branch, this model may not necessarily be compliant with the desired standardization.

1.2. Hypothesis

The hypothesis of this thesis is that based on historical information stored in event logs and in a configurable process model it is possible to obtain a configuration that allows the derivation of a process model automatically using Process Mining techniques.

1.3. Objectives

There are three general objectives of this thesis, which are broken down into the specific objectives presented below.

1.3.1. General objectives

This thesis presents an approach with a threefold objective. First, we propose an additional use case to those presented by van der Aalst (2012), where we combine a configurable process model (manually or automatically generated (Buijs et al., 2013)) and an event log to derive a process model that better represents the observed behavior in the historical data, depicted in Figure 1.1 d) as *ConCMEV*. Second, to support this use case extension, we redefine the configurable process model representation, in particular how to represent configurable process trees (Schunselaar et al., 2012), so as to generalize and simplify the description of the configuration process. Third, we propose a derivation framework that receives as input a configurable process model and an event log; as depicted in Figure 1.2[2], both inputs are part of the extended configurable process model use case.

1.3.2. Specific objectives

The specific objectives of this research are:

(i) to conceptualize the problem in order to be able to propose solutions that are applicable and allow us to generate satisfactory results.

(ii) to develop derivation strategies that explore configuration alternatives of a configurable process model using a configurable process model.

[2]Three alternative derivation strategies allow to obtain a configuration that later on is used to derive a process model from the configurable process model.

(iii) to implement derivation strategies automatically that allow finding a configuration to be applied to a configurable process model.

(iv) to verify the proposed strategies using controlled experiments of a realistic data set and finally to validate the strategies using a real-life data set.

1.4. Methodology

To address these objectives, we apply the following methodology:

- Literature review of state-of-the-art research in the field of derivation of process models, tools used and how derivation of process models is performed.
- Conceptualization of the problem that allows us to propose different strategies that solve the problem of automatic derivation of process models using an event log and a configurable process model.
- Development of derivation strategies. We consider three alternative strategies:
  - Exhaustive strategy that evaluates all possible configurations in a configurable process model to finally select the optimal configuration.
  - Genetic strategy that has as a representation of a configuration that can be used for the evolutionary process. The aim of this strategy, based on an evolutionary algorithm, is to converge to a final optimal configuration.
  - Greedy that allows to look for local solutions with the intent of finding a good final solution.
- Verification and validation of proposed strategies using:
  - Realistic case study based on a university academic programming software. This experimental set will allow us to verify if the strategies produce expected results.
  - Real case study based on the building permits process of five Dutch municipalities. The purpose of applying this experimental set is to validate the behavior of the proposed strategies, where our results will be compared with those of Buijs (Buijs, 2014b).

1.5. Proposed derivation framework

The proposed framework incorporates three derivation strategies: an *exhaustive* method, used as a reference approach that finds an optimal configuration in a wide search space, a *genetic* evolutionary method, designed as a smart technique that evolves until it finds a good configuration, and a *greedy* method, designed as a heuristic to find a satisfying configuration in a short computing time. The configuration obtained by any of these three strategies is then applied to the configurable process model in order to derive a process model. Additionally, we have tested the feasibility and applicability of the framework using two different sets of experiments: an educational process and a real-life municipality scenario.

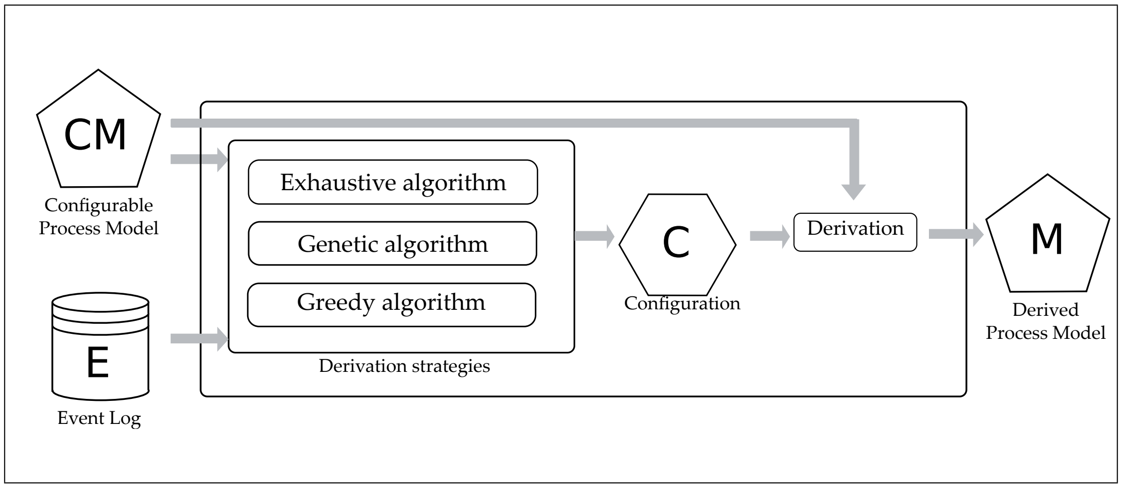


FIGURE 1.2. Overview of the proposed framework to derive a process model.

1.6. Organization of the thesis

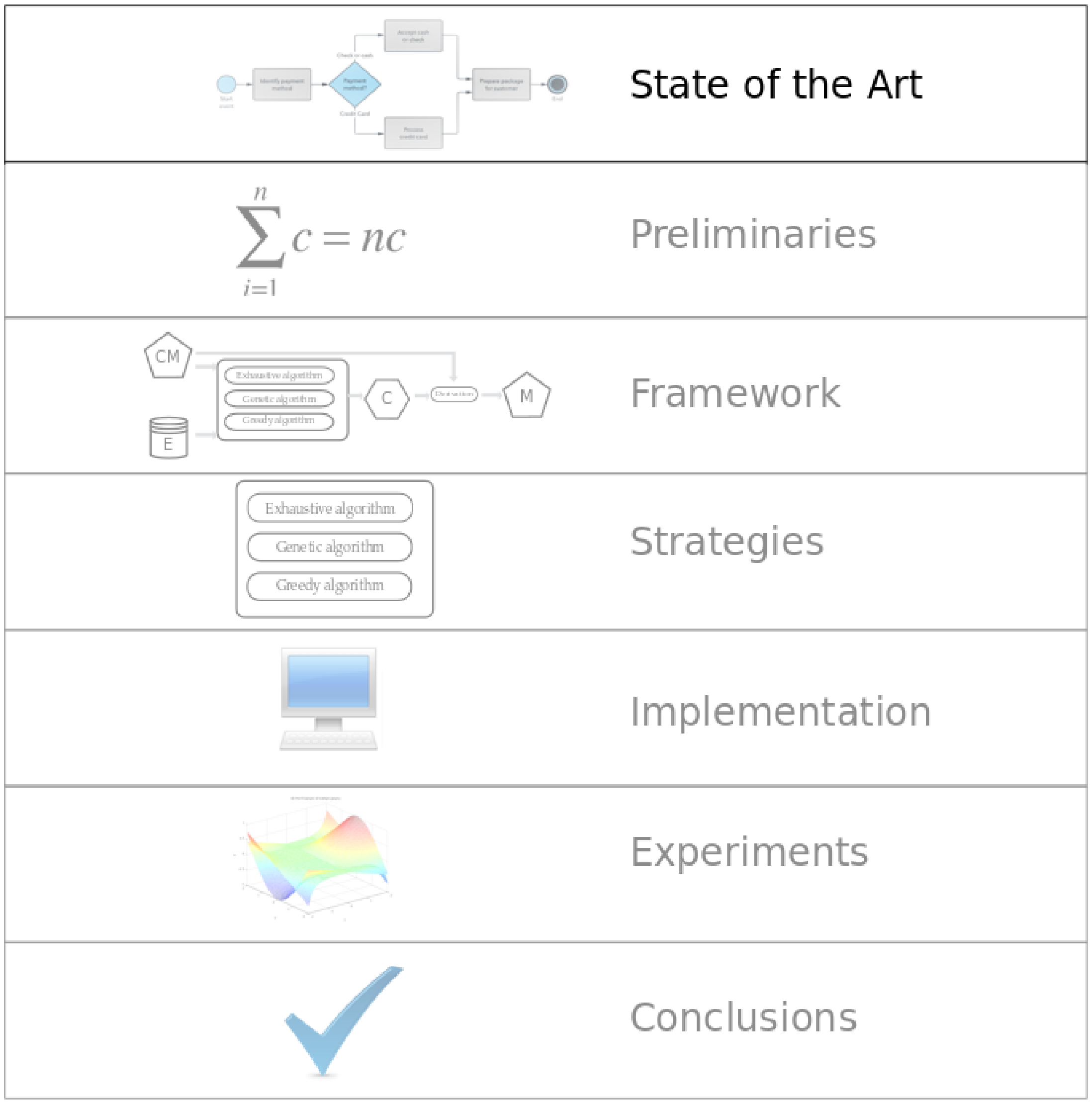
This thesis is organized as follows: state of the art is presented in chapter 2, where we also introduce a new use case in which we based this thesis. Later, chapter 3 introduces the theoretical foundation of the proposed framework. In chapter 4, we present the framework. The three strategies (exhaustive, genetic, and greedy) that allow finding the best configuration in order to derive a process tree from a configurable process tree that better represents the observed behavior in an event log are presented in chapter 5. The implementation of the framework is described in chapter 6. Results and discussions are presented in chapter 7. Finally, conclusions and future work are presented in chapter 8.

The results of this thesis have been partially published in (Arriagada-Benítez, Sepúlveda, Munoz-Gama, & Buijs, 2017):

Strategies to Automatically Derive a Process Model From a Configurable Process Model Based on Event Data
Mauricio Arriagada-Benítez, Marcos Sepúlveda, Jorge Munoz-Gama and Joos C. A. M. Buijs
Applied Science 2017, 7, 1023.

2. STATE OF THE ART

This chapter shows the state of the art of configurable process models through a literature review.



Large volumes of information are generated on a daily basis. One of the reasons for this exponential increase in data has been the adoption of new information technologies that allow not only to generate, but also to store and manage, data through information systems. In many cases, information is systematized through models that allow capturing data from a specific domain, understanding that a model is an abstract representation of reality that allows us to describe and understand the functioning of a system. The use of models facilitates the study of a system. Some of the advantages of using a model include the low cost of implementation, the time efficiency of modeling, and the opportunities offered by simulation. It is possible to classify different types of models depending on the study area. For example, there are models designed in some study fields such as statistics, mathematics, physics or economics.

In business process management, much of the recorded data is obtained from a series of activities that are carried out during the process. Dumas, Van der Aalst, and Ter Hofstede (2005) point out that there are specialized software systems managing not only activities but also processes such as ERP (Enterprise Resource Planning) systems, BPM (Business Process Management) systems, and CRM (Customer Relationship Management) systems. Nowadays, many companies are using process models to implement their information systems. This is because process models represent not only the activities that are performed in a process, but also the interactions between those activities. This type of model allows, first, to describe a business process which is made up of a logically related set of activities that uses the resources of the organization, then, to provide defined results, and finally, to achieve the objectives of the business.

According to van der Aalst (2016), the process models are used for different purposes: insight, discussion, documentation, verification, performance analysis, animation, specification and configuration. This research focuses on this last topic, the configuration of configurable process models.

2.1. Configurable process models

Today, it is possible to find that the same business process runs differently in different business divisions. These variants are presented in the same business process that is performed in different geographic areas, such as retail premises, or even in different areas of work within the same location, such as faculties of the same university. The challenge for organizations is to maintain a degree of flexibility and standardization of their business processes.

A configurable process model is a representation of possible variants of a business process that can be visualized in a single model (van der Aalst, Dreiling, Gottschalk, Rosemann, & Jansen-Vullers, 2005; Gottschalk, van der Aalst, & Jansen-Vullers, 2007; Gottschalk, Wagemakers, Jansen-Vullers, van der Aalst, & Rosa, 2009). These variants represent the best practices of how the process should be performed (Rosa, Dumas, ter Hofstede, Mendling, & Gottschalk, 2008).

A configurable process model presents points of variation on which it is possible to apply a certain configuration: enable, hide and block tasks (van der Aalst, Lohmann, & Rosa, 2012). These configuration points allow to select variants of the configurable model. After configuring the variation points of a model, a derived model is obtained.

The utility of integrated representation of multiple variants is the flexibility provided by the configurable model to derive process models, where each derived model represents a possible way of executing the process. Finally, having a configurable process model facilitates not having to redefine a business process model that is repeated several times within the organization (Rosa, Dumas, ter Hofstede, & Mendling, 2011).

According to Seidel, Rosemann, Hofstede, and Bradford (2006), the importance of having a configurable business model is that it allows us to identify how the same process could be executed in different scenarios. As an example, we can mention the scope of municipalities, where despite of the same regulations, some processes are executed differently in different municipalities. In some cases the differences in the models are reasonable variants that are a consequence of unique characteristics of municipalities, for example, large municipalities, rural municipalities or municipalities with fewer resources. In other cases, the differences between models reflect bad practices. Therefore, it would be beneficial to have a model with all reasonable variants of how you can run municipality business processes.

Based on configuration points on a configurable process model, it is possible to derive a process model that satisfies business needs for a given division. In the past, this derivation was done manually either directly from a configurable process model using process modeling tools specially adapted for this purpose, or using questionnaires that allowed the derivation to be made in a more intuitive way. The difficulty of using those two techniques was that obtaining a process model from a configurable process model required the knowledge of domain experts. The desire to resolve this difficulty was the main drive behind this research, which proposes strategies to derive a process model automatically by finding the best configuration based on a event log.

Nowadays, not only data scientists, but also professionals from different fields work on strategies to obtain better results of processing and visualizing large volumes of information. For this analysis, where events are being recorded as part of processes that take place in organizations, there is a discipline called process mining, whose purpose is to use event data to extract information related to processes.

2.2. Process Mining

Process Mining is a new discipline that integrates machine intelligence, data mining, modeling and process analysis, which allows to discover, monitor and improve organizations' business processes by extracting knowledge from event data (event log, see Chapter 3) available in corporate information systems (van der Aalst, 2016). The extraction of knowledge from event logs allows to discover models and analyze to what extent the way in which a process is performed in reality corresponds with what is defined in the preexisting models. Some techniques for process modeling in Process Mining are presented in

Tiwari, Turner, and Majeed (2008); van der Aalst et al. (2003); van der Aalst and Weijters (2004).

There are three types of Process Mining. The first type is process discovery, in which models are constructed to reflect the log of events of a business process. Some of the algorithms used are Alpha Miner (van der Aalst, Weijters, & Maruster, 2004), represented by Petri nets; Heuristic Miner (A. Weijters, Aalst, & A K Medeiros, 2006), Flexible Heuristic Miner (A. J. M. M. Weijters & Ribeiro, 2011), whose representation are causal networks (Annex B); and Fuzzy Miner (Günther & van der Aalst, 2007). (van der Aalst, Dumas, Ouyang, Rozinat, & Verbeek, 2008) where fitness quality measure was introduced by Rozinat and van der Aalst (2005, 2008) to assess the discovered process model. The second type is to verify if the actual business process and information systems are aligned. In the research of Rozinat and van der Aalst (2008), the quality of a process model is measured with respect to the observed behavior as recorded in an event log. The last type is enhancement, where the main task is to extend or improve a process model with additional information that can be found in an event log. In this area, performance analysis could be used to identify bottlenecks, frequencies, and service levels (van der Aalst, 2016).

Process mining as a discipline of study has been applied in different domains: analyzing treatment processes in hospitals, process analysis in municipalities, improving customer service processes in multinational corporations, understanding the browsing behavior of customers, among others (van der Aalst, 2016).

2.3. Current Use Cases

Over time, several papers on the Business Process Management field have been presented at BPM conferences. van der Aalst conducted a research where he analyzed papers sent to BPM conferences from 2003 to 2011 (van der Aalst, 2012). He identified the twenty use cases shown in Figure 2.1. Some use cases involved the participation of people in the process. For example, the use case Design Model (DesM) is related to the creation of a process model by a person. Alternatively, the use case Discover model from event data

(Disc) represents how a model can be created automatically based on event data, for example, using process mining techniques. If a model is created by the use case DesM, then the model is Descriptive (D), Normative (N), and/or Executable (E). A model discovered through process mining (DiscM), on the other hand, is typically not normative since it is based on observed behavior (it can be D, and eventually E).

Of the 20 use cases identified by van der Aalst (2012), use cases related to configurable models are: Design Configurable Model (DesCM), Merge models into Configurable Model (MerCM), and Configure Configurable Model (ConCM). The two most frequently used use cases are design model (DesM) and enact model (EnM) since they are for general purposes, whereas the use cases related to configurable models have low use frequency, since they are very specific.

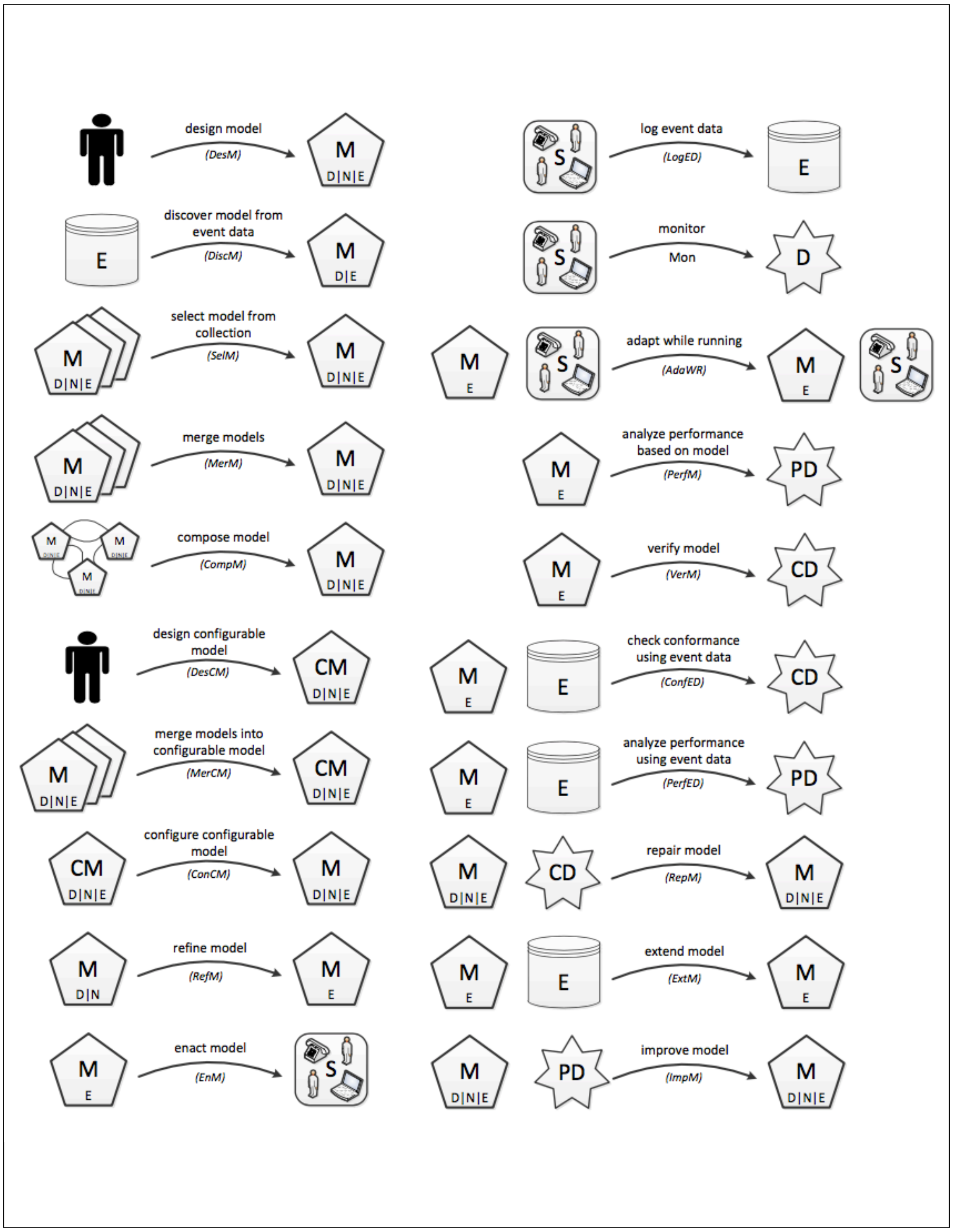


FIGURE 2.1. Use cases presented at BMP conference in 2015.

(van der Aalst, 2012)

It is possible to compose a use case by chaining other use cases. Thus, in this thesis we propose a new use case, called Configure Configurable Model based on Event Log (ConCMEV) which considers a configurable process model and a event log as input and a derived process model as output (see Figure 1.1). Although it is based on observed behavior, the derived process model is restricted to the processes that can be derived from the configurable model. Therefore, the obtained model is an interesting trade-off between a normative model (N) and a descriptive (D) one.

During his research, van der Aalst also found that despite of the good quality of the set of 289 BPM papers analyzed, some weaknesses could be identified in all of them. This information can be found in Table 8.1, in chapter Conclusions.

2.4. Related work

The majority of research in the area of configurable process models has addressed the issue of describing a configurable process model, or the issue of manually obtaining such a configurable process model (Rosa, Dumas, Uba, & Dijkman, 2010; Mendling & Simon, 2006; Gottschalk, 2009; Rosa, Dumas, Uba, & Dijkman, 2013; Gottschalk, van der Aalst, & Jansen-Vullers, 2008a; Schunselaar et al., 2012). Manual process model configuration (i.e., configuration performed by the user) has been addressed by Schunselaar et al. (2012); Rosa, van der Aalst, Dumas, and ter Hofstede (2009). In Rosa et al. (2009), for example, a questionnaire-driven approach for configuring a reference model is taken, guiding the user in defining a configuration. The work by Schunselaar et al. (2012) uses a configurable tree-like representation which is sound by construction. Applied in the Configurable Services for Local Governments (CoSeLoG) project (Buijs, 2014a), this approach merges variants of different municipalities to create a configurable process model. The same author underlines in (Schunselaar et al., 2014) the difficulty of creating a configuration since the user needs a high abstraction level about the process. Hence, the author uses a meta-model to automatically construct an abstraction that helps the end user to apply configurations. The same author has also extended the work to consider several qualitative process aspects such as performance, cost, and satisfaction indicators. The results are then presented to the end user, who inspects the proposed configurations and selects one to be applied.

A configurable process model allows a reference model to be adapted to different business scenarios where the process is being executed (e.g., different branches of a large organization or different companies belonging to a corporate group). A complementary approach aims at adapting a general process model to different modeling views that are needed for the integrated modeling of business processes and information systems (Becker, Delfmann, & Knackstedt, 2007; Becker, Delfmann, Dreiling, Knackstedt, & Kuropka, 2004).

Event data has been used to create process models through the use of different techniques. For example, the research driven by van Oirschot (2014) uses an event log to discover a process model based on the observed behaviour of traces using clustering technique and a tree-like representation. However, event data is not often used to configure a configurable process model. One of the few approaches using this is the Evolutionary Tree Miner (Buijs, 2014b), which is able to discover a process model including configurations, when given multiple event logs. Nevertheless, the performance of the Evolutionary Tree Miner is not optimal. Because it is an evolutionary algorithm, and the configuration aspect adds many possibilities, the task of discovering a configurable process model and several configurations becomes challenging. Moreover it is not possible to extend it to the use case proposed in this thesis.

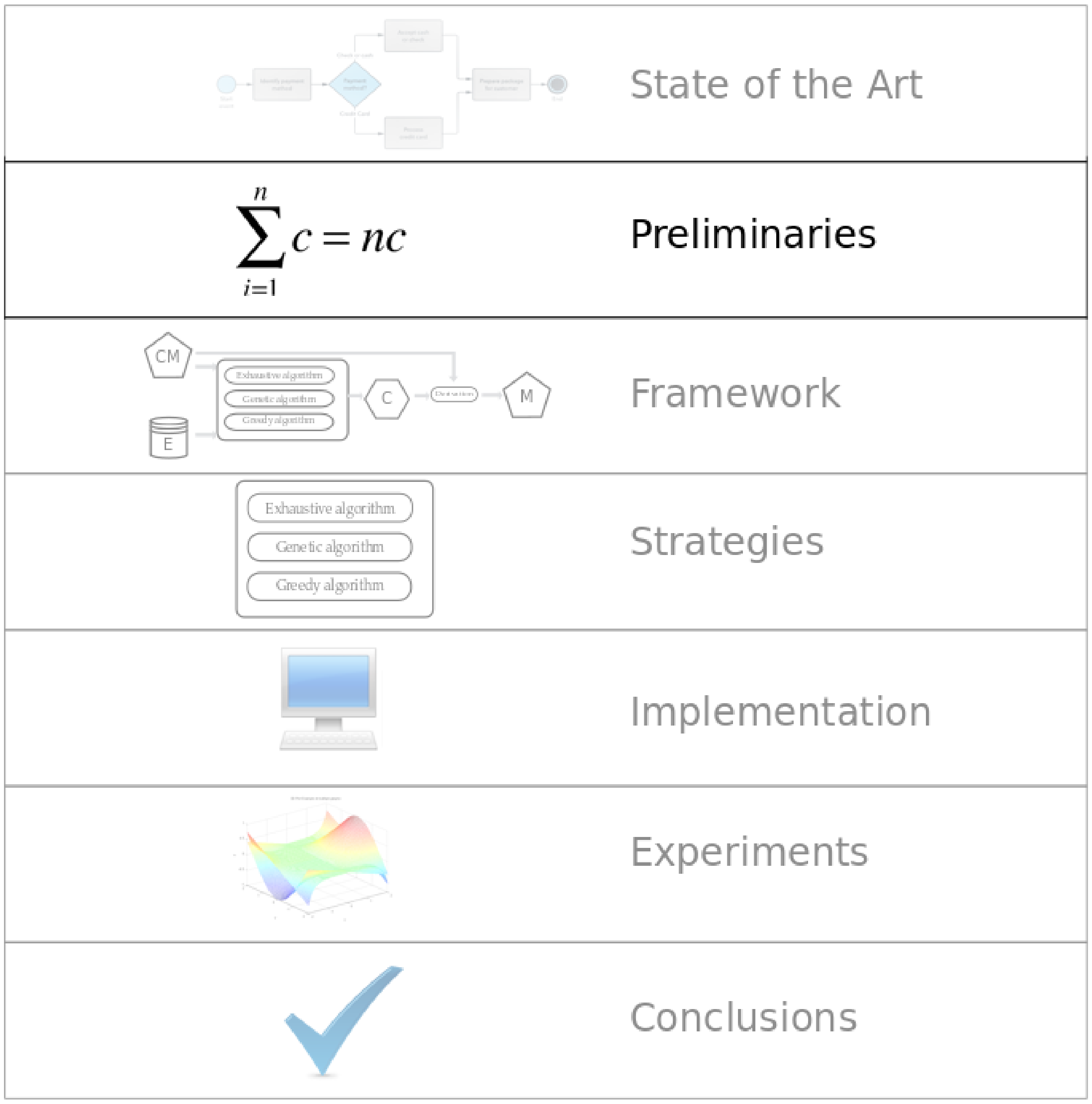
The main limitation of existing approaches is the restricted number of configurations a configurable process model can have. The approach proposed in this thesis aims to enhance the work already done in configurable process models by allowing a large number of configurations. This research also provides a framework that allows to combine a configurable process model and an event log to derive a process model that better represents the observed behavior in the event log.

New investigations related to models of configurable processes are oriented on how to create a better creation of these. In the investigation of (Derguech, Bhiri, & Curry, 2017) only the configurable process model is created, but the next step that is the configurations is not addressed

Similar to existing approaches, we implemented our techniques in the open source process mining tool *ProM*.

3. PRELIMINARIES

In this chapter, we introduce the theoretical foundation of the proposed framework, such as event log, process tree, configurable process tree, and how to measure the quality of a derived process tree to represent the observed behavior in an event log.



Our proposal is based on obtaining the best configuration delivered by each of the strategies we propose. To achieve this, we formally present what is the data that our strategies use such as set, multiset, event log, and process tree.

3.1. Set, Multiset, Sequence, and Concatenation

A *multiset* (or a bag) is a generalization of the concept of *set*, where its elements may appear multiple times. For a given set *A*, $\mathcal{B}(A)$ is the set of all multisets over *A*. For a multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times the element $a \in A$ appears in b. For example, $b_1 = [\,]$, $b_2 = [x, x, y]$, $b_3 = [x, y, z]$, $b_4 = [x, x, y, x, y, z]$, $b_5 = [x^3, y^2, z]$ are multisets over $A = \{x, y, z\}$. b_1 is the empty multiset, b_2 and b_3 both consist of three elements, and $b_4 = b_5$ since the order of the elements is irrelevant; b_5 representation is preferred because it is a more compact way of representing the same elements. Note that sets are written using curly brackets while multisets are written using square brackets.

For a given set *A*, A^* is the set of all finite sequences over *A*. A finite *sequence* of length *n*, $\rho = \langle a_1, a_2, a_3, ..., a_n\rangle \in A^*$, is a mapping $\{1, ..., n\} \to A$. Its length is denoted by $|\rho| = n$ and the element at position i (a_i) is denoted as ρ_i. Also, $\langle\,\rangle$ is the empty sequence. Note that sequences are written using angle brackets. For two sequences, ρ_1 and ρ_2, $\rho_1 \cdot \rho_2$ denotes the concatenation of two sequences. For example, $\langle a, b, c\rangle \cdot \langle m, n\rangle = \langle a, b, c, m, n\rangle$.

3.2. Event Log

Information systems record event data in the form of event logs that register events related to the execution of processes within an organization. Each event is identified as part of a trace (a process instance) that is executed for a given process.

Definition 3.1 (Trace, Event log)**.** *Let A be a set of activities over a universe of activities. A* trace $\sigma \in A^*$ *is a sequence of activities.* $L \in \mathcal{B}(A^*)$ *is an* event log, *i.e., a multiset of traces.*

For instance, $\langle a, b, c, e, g\rangle$ is a trace that belongs to an event log $L_1 = [\langle a, b, c, e, g\rangle^3, \langle a, c, b, e, g\rangle^4, \langle a, d, f, g\rangle^2]$.

Definition 3.2 (Projection)**.** *Let A be a set and $A' \subseteq A$ one of its subsets. $\sigma\restriction_{A'}$ denotes the projection of $\sigma \in A^*$ on A', e.g., $\langle a, a, b, c\rangle\restriction_{\{a,c\}} = \langle a, a, c\rangle$. The projection can also be applied to multisets, e.g., $[x^3, y, z^2]\restriction_{\{x,y\}} = [x^3, y]$.*

Projection can be used to obtain a sublog of an event log. For instance, $L_1\restriction_{\{a,e,g\}} = [\langle a, e, g\rangle^7, \langle a, g\rangle^2]$.

As an example of an event log, we present the Table 3.1 that contains information of case id, activity name, timestamp, and role.

TABLE 3.1. Sample of an event log

case id	activity	user	timestamp	role
1	a	user 1	21-05-18 14:12	receptionist
1	b	user 12	21-05-18 14:32	analist
1	d	user 5	21-05-18 15:01	analist
1	e	user 4	21-05-18 15:09	technician
1	f	user 9	21-05-18 15:11	technician
1	g	user 3	21-05-18 15:18	cashier
1	h	user 2	21-05-18 15:23	dispatcher
2	a	user 1	21-05-18 15:31	receptionist
2	b	user 5	21-05-18 15:47	analist
2	d	user 12	21-05-18 15:50	analist
2	g	user 4	21-05-18 15:57	technician
2	e	user 4	21-05-18 16:05	technician
2	f	user 3	21-05-18 16:15	cashier
2	h	user 2	21-05-18 16:20	dispatcher
3	a	user 1	21-05-18 16:33	receptionist
3	b	user 4	21-05-18 16:35	technician
3	c	user 2	21-05-18 16:46	dispatcher

3.3. Process Tree

Playing an important role in organizations, a process model can be used to represent a workflow task execution in a certain process (van der Aalst, 2016). The use of Petri nets as modeling notation is common in both Discrete Event Systems and Process Mining literature (van der Aalst, 2016). In our framework we use the different, but still related, process tree notation to represent a process, similar to other approaches in configurable process models literature (Buijs, 2014b). A *process tree* (van der Aalst, Buijs, & van Dongen, 2011; Buijs, 2014b) is a tree-structured process model, where the leaf nodes represent the activities, and the non-leaf nodes represent control-flow operators, e.g., sequence ($\rightarrow$), exclusive choice ($\times$), inclusive choice ($\vee$), parallelism ($\wedge$) and loop ($\circlearrowleft$). A silent activity is denoted by τ and cannot be observed; it is used to model processes where an activity can be skipped under some specific circumstances.The process tree notation ensures soundness, and it is used by a wide range of process mining techniques, such as Evolutionary Tree Miner (Buijs, van Dongen, & van der Aalst, 2012b), Inductive Miner (Leemans, Fahland, & van der Aalst, 2013a) and Inductive Miner-infrequent (Leemans, Fahland, & van der Aalst, 2013b). Its formal definition is as follows:

Definition 3.3 (Process tree). *Let A be a finite set of activities, with $\tau \notin A$ representing a silent activity. $\oplus = \{\rightarrow, \times, \vee, \wedge, \circlearrowleft\}$ is the set of* process tree operators. *A* process tree *is recursively defined as follows (van der Aalst, 2016):*

- *if $a \in A \cup \{\tau\}$, then $Q = a$ is a process tree,*
- *if $Q_1, Q_2, \ldots, Q_n$ are process trees where $n \geq 1$, and $\oplus \in \{\rightarrow, \times, \vee, \wedge\}$, then $Q = \oplus(Q_1, Q_2, \ldots, Q_n)$ is a process tree, and*
- *if $Q_1, Q_2, \ldots, Q_n$ are process trees where $n \geq 2$, then $Q = \circlearrowleft (Q_1, Q_2, \ldots, Q_n)$ is a process tree.*

The nodes of a process tree, both operator and activity nodes, are denoted as $N(Q)$.

Notice that both Petri net and process tree modeling notations are closely related, and that conclusions obtained in one model can be easily extrapolated to the other. Moreover, van der Aalst (2016) presents a mapping between process tree and Petri net-based workflow nets, and it can be easily adapted for other representations such as BPMN, YAWL, EPCs, among others (van der Aalst, 2016). However, process trees also preserve interesting properties for the analysis and the verification, such as soundness by construction, and the block-structured (van der Aalst, 2016).

Figure 3.1 shows an example of a process tree model Q_1 that contains 7 activities and 5 operators. This process tree contains a sequence operator ($\rightarrow$) as a root node, i.e., its branches will be executed from left to right. Hence, the first activity to be executed will be a. Then, there is a loop operator ($\circlearrowleft$). Its leftmost branch represent the **do** part of the loop, it will be executed at least once and the loop execution will always starts and ends with it. In this case, there is an exclusive choice operator ($\times$) in the leftmost branch, indicating that either the activity b or the parallel ($\wedge$) activities c and d will be executed. The rightmost branch of the loop operator ($\circlearrowleft$) represents the **redo** part of the loop, which in this case contains only the activity e. The process ends with a exclusive choice operator ($\times$) that indicates the final activity will be f or g.

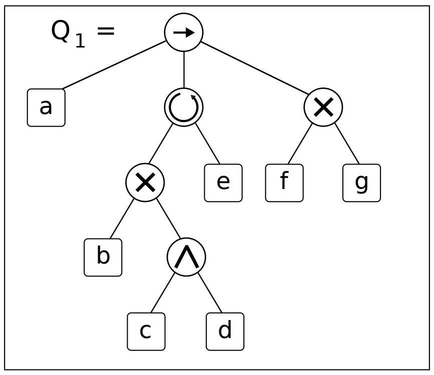


FIGURE 3.1. Example of a process tree model.

Assessing the quality of a process model is very challenging and is characterized by four different dimension: fitness, simplicity, precision, and generalization.

3.4. Quality Metrics

Several configurations can be applied to a configurable process tree. In order to assess the quality of a derived process tree to represent the observed behavior in an event log, a quality metric must be defined. Quality is usually measured considering a trade-off among the following four quality criteria: *fitness, precision, generalization, and simplicity* (van der Aalst, 2016; Buijs et al., 2012b; Munoz-Gama, Carmona, & van der Aalst, 2013).

Fitness Fitness quantifies the extent to which the behavior of the event log can be replayed in the process model. There exist various methods to calculate the fitness metric. The most common known methods are: Alignment-based (Adriansyah, Munoz-Gama, Carmona, van Dongen, & van der Aalst, 2012) replay and Token-based (Rozinat & van der Aalst, 2005) replay. In this thesis we used the method based on alignment since it works better on process tree models.

Precision Precision indicates how much additional behavior the process model allows that is not seen in the event log. However, enumerating all possible traces of the process model is not feasible for larger process models, especially when they have many parallel activities. Moreover, in case of loops, the allowed behavior is infinite. Therefore estimations of the allowed behavior of a process model need to be made (Buijs, 2014b).

Generalization Generalization estimates how well the process model describes the behavior of the (unknown) process, and not only the event log with the observed system behavior. Thus, a process model should generalize and not restrict behavior. A process model that does not generalize is “overfitting”. Overfitting is the problem that a very specific model is generated whereas it is obvious that the log only holds example behavior, i.e., the model explains the particular sample event log, but a next sample event log of the same process may produce a completely different process model. Process mining algorithms need to strike a balance between “overfitting” and “underfitting”. A model is overfitting if it does not generalize and only allows for the exact behavior recorded in the log. This means that the corresponding mining technique assumes a very strong notion of completeness: “If the sequence is not in the event log, it is not possible!”. An underfitting model over-generalizes the things seen in the event log, i.e., it allows for more behavior even when there are no indications in the event log that suggest this additional behavior (van der Aalst, 2016).

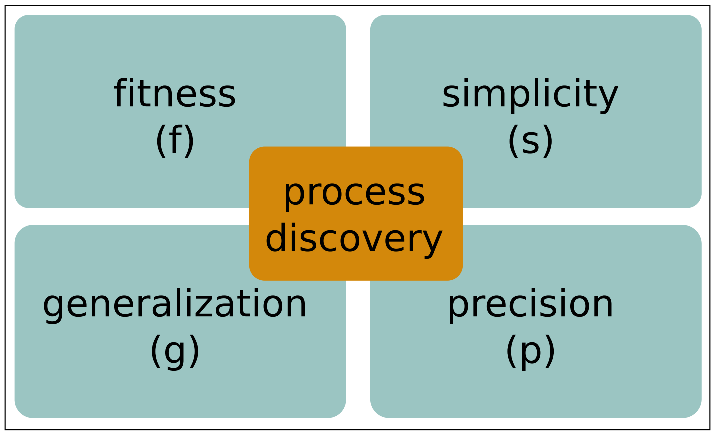
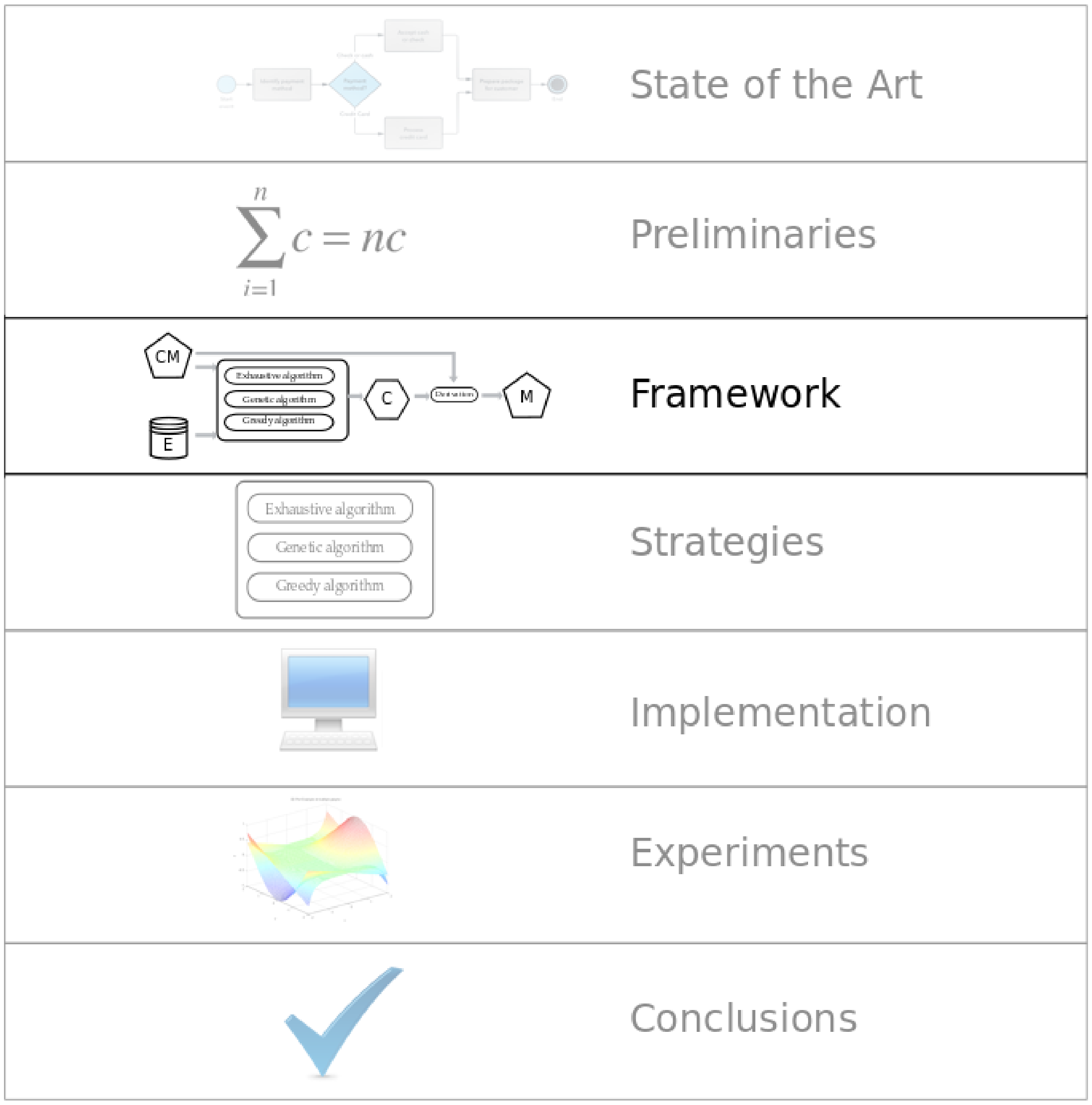


FIGURE 3.2. Quality metrics of a process model.

Simplicity The quality dimension of simplicity quantifies the simplicity of the process model and therefore is the only quality dimension that is not necessarily related to the behavior of the process model or event log. Simplicity of the process model is defined by two aspects. The first aspect is related to Occam's Razor, which states that one should not increase, beyond what is necessary, the number of entities required to explain anything. Since the other three quality dimension already evaluate what is necessary in the process model, simplicity mainly focusses on reducing the size of the process model. The second aspect that can be considered in the simplicity dimension is the simplicity of the process model as perceived by a user. However, this aspect is hard to capture and measure, since this is related to the understandability of the process model.

4. FRAMEWORK

In this chapter we present the general framework used to develop our strategies based on the chained use case we proposed.



Our framework receives a log and a configurable process tree as input. A process tree was already presented in chapter 3. We introduced the basis of configurable process tree. However we have not given a formal definition of a configurable process tree.

4.1. Configurable Process Tree

Extensive work on representing configurable process models as process trees has been addressed by Buijs et al. (2013); Schunselaar et al. (2012). We have slightly redefined the definition of configurable process trees described by them on our proposed framework, in order to allow greater flexibility and expressiveness in the configurable nodes. A configurable process tree (CPT) represents a family of process models (Rosa et al., 2011). All processes of a family share the same tree topology, while differences are handled using configurable nodes. Applying a particular configuration to these configurable nodes produces a variant of the configurable process model, a derived process model. A configurable node can be set to enable (or allow), hide or block. Enable (allow) means the node is enabled to be visited, hide means the node can be skipped over, and block means the node cannot be reached. Foundation of configurable process tree is described in Schunselaar et al. (2012), and also applied in Buijs et al. (2013).

Definition 4.1. (Process tree configurators)
$\ominus = \{H, B, E\}$ *is the set of* process tree configurators*, and* $\ominus_\times = \{\{H\}, \{B\}, \{E\}, \{H, B\}, \{H, E\}, \{B, E\}, \{H, B, E\}\}$ *is the set of all subsets of the* process tree configurators*, where:*

- *H: $hide$ a node. It makes a node unobservable, replacing it by a τ node.*
- *B: $block$ a node. It makes the leading path to this node unreachable. When blocking a node, several cases might occur; details can be found in (Buijs, 2014b).*
- *E: $enable$ a node. It essentially allows a node to be performed, either an operator or an activity, so as it behaves normally.*

Definition 4.2. (Configurable process tree)
A configurable process tree $Q^\alpha = (Q, \alpha)$ *is comprised of a process tree* Q *with* $N(Q)$ *nodes, and a partial* configuration function $\alpha : N(Q) \nrightarrow \ominus_\times$ *defining a configuration set for some nodes.* $N^\alpha(Q^\alpha) \subseteq N(Q)$ *is the set of* configurable nodes, *i.e.,* $N^\alpha(Q^\alpha) = domain(\alpha)$*. For the sake of clarity, let us assume an ordering among the configurable nodes, i.e.,* $n_1, n_2, \ldots, n_{|N^\alpha(Q^\alpha)|}$*.* $C(Q^\alpha)$ *is the set of all possible* configurations *of* Q^α*, i.e.,* $C(Q^\alpha) = \{\langle c_1, c_2, \ldots, c_{|N^\alpha(Q^\alpha)|}\rangle | c_i \in \alpha(n_i)\}$.

Figure 4.1 is an example of a configurable process tree Q_1^α that contains 4 configurable nodes, listed from 1 to 4. Configurable nodes 1, 3 and 4 can be either *hidden* or *enabled*, whereas the configurable node 2 can be either *blocked* or *enabled*.

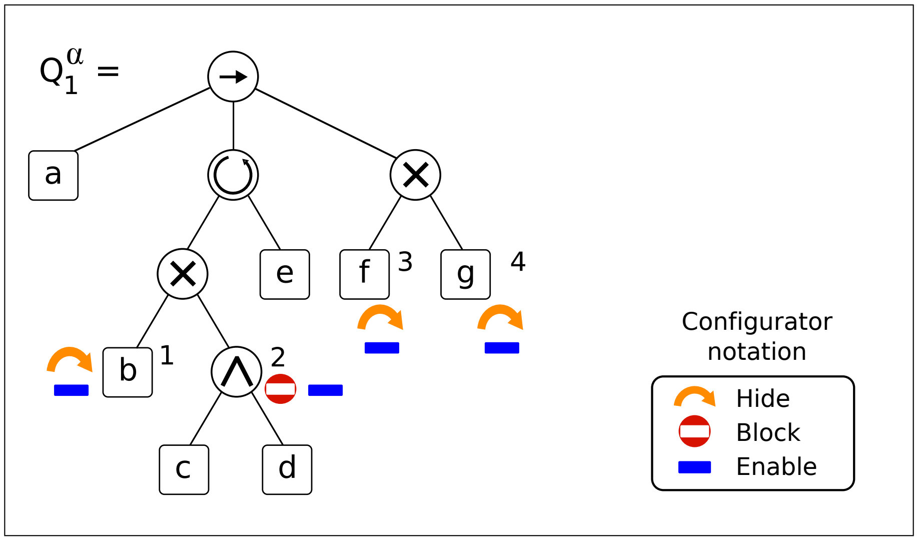


FIGURE 4.1. Example of a configurable process tree containing 4 configurable nodes.

It is possible to apply a configuration to a configurable process tree in order to obtain a process tree, known as a derived process tree, as follows:

Definition 4.3. (Derived process tree)
Let Q^α *be a configurable process tree, and let* $c \in C(Q^\alpha)$ *be a possible configuration.* $derive(Q^\alpha, c) = Q_c$ *is the function that generates a derived process tree* Q_c *from* Q^α *by applying the configuration* c*, using the rules defined in (Buijs, 2014b).*

Figure 4.2 presents a running example of the execution of the derivation framework shown in Figure 1.2. A configurable process tree and an event log are the inputs, depicted in a) and b), respectively. The implemented derivation strategies use a common data structure to represent all the feasible configurations for all configurable nodes, shown in c). Each derivation strategy allows to obtain a configuration, shown in d), which is then used to derive a process tree, shown in e). Figure 4.2 also depicts the notation for the process tree configurators and the model activities in f) and g), respectively.

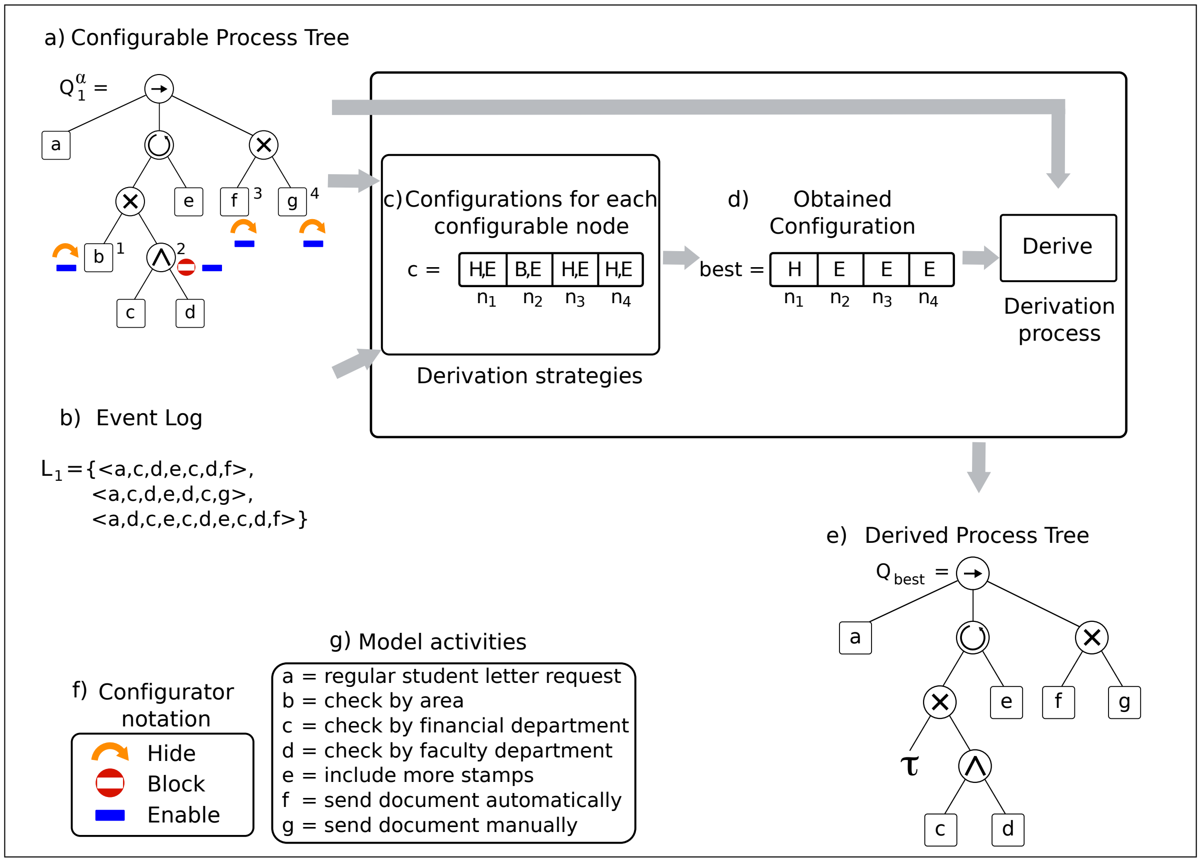

a) Configurable Process Tree
Q_1^α =
a
e
f 3
g 4
b 1
2
c
d
c) Configurations for each configurable node
c = H,E B,E H,E H,E
n1 n2 n3 n4
Derivation strategies
d) Obtained Configuration
best = H E E E
n1 n2 n3 n4
Derive
Derivation process
b) Event Log
L1 = {<a,c,d,e,c,d,f>, <a,c,d,e,d,c,g>, <a,d,c,e,c,d,e,c,d,f>}
e) Derived Process Tree
Q best =
a
e
f
g
τ
c
d
g) Model activities
a = regular student letter request
b = check by area
c = check by financial department
d = check by faculty department
e = include more stamps
f = send document automatically
g = send document manually
f) Configurator notation
Hide
Block
Enable

FIGURE 4.2. Running example that illustrates the derivation framework.

Conformance checking is a subdiscipline of process mining that allows to compare the behavior allowed by a process model with the behavior recorded in an event log to find commonalities and discrepancies, and also to compute metrics for each of the four quality criteria (Munoz-Gama, 2016).

Definition 4.4. (Conformance)
Let Q be a process tree and let L be an event log. Let f, p, g, s be the fitness, precision, generalization, and simplicity metrics as defined in (Buijs, 2014b) with a range [0,1], being 1 the target value, and let $W = (w_f, w_p, w_g, w_s)$ be the weights given to each metric, respectively. Conformance *is defined as:*

$$conformance(Q, L, W) = \frac{f(Q, L) \cdot w_f + p(Q, L) \cdot w_p + g(Q, L) \cdot w_g + s(Q, L) \cdot w_s}{w_f + w_p + w_g + w_s} \tag{4.1}$$

In this article, we have defined the conformance metric based on (Buijs, 2014b) and used the corresponding implementation to evaluate the quality of a given process model. However, the proposed framework is generic and independent of the conformance metric used.

An optimal configuration is the one that allows to obtain a derived process tree that better represent a given event log.

Definition 4.5. (Optimal configuration)
Let Q^α be a configurable process tree, let L be an event log, and let W be the weights of the quality metrics. A configuration $c \in C(Q^\alpha)$ is an optimal configuration *if and only if there is not another configuration $c' \in C(Q^\alpha)$ such that $conformance(derive(Q^\alpha, c'), L, W) > conformance(derive(Q^\alpha, c), L, W)$.*

Our goal is to automatically obtain a configuration to derive a process tree from a configurable process tree that maximizes the *conformance* function for a given event log. To accomplish this goal, we have implemented three strategies, which are detailed in the next section.

4.2. Proposed framework

To derive a configuration, we proposed three different strategies. We formally present them in chapter Framework. The framework has two inputs: an event log and a configurable process tree. The output of the framework is a derived process model. The big box of Figure 1.2 shows the internal procedure to obtain a configuration through one of the strategies.

The usage of the framework depends on the option that the user set to derive a process model.

There are couple of steps to follow in order to derive a configuration an thus obtain a process model:

(i) The user inputs a configurable process tree and an event log.
(ii) The user has to select one of the three implemented strategies.
(iii) The framework obtains the best configuration based on the selected strategy.
(iv) The framework applies the configuration to the configurable process tree to obtain a process model that represent such configuration.

5. DERIVATION STRATEGIES

In this chapter, we present the three strategies we implemented to derive a configuration out of a configurable process tree and an event log.

As part of the proposed framework, we have designed three different derivation strategies that are able to find a suitable configuration in order to derive a process tree. The first strategy is based on an exhaustive approach, guaranteeing to find an optimal configuration. The other two strategies are based on heuristics that find a configuration that allows to derive a reasonably good process tree in a faster time. Each of these strategies is described hereafter. Notice that the approach proposed in this article is different to the one proposed in (Buijs, 2014b). In (Buijs, 2014b), the input is a collection of event logs and the output is a configurable process model. In our case, the inputs are an already existing configurable process model and an event log. The output is a feasible configuration of the input configurable process model, so as the derived process model obtained with such a configuration better represents the input event log.

5.1. Obtaining a configuration based on the exhaustive strategy

The first strategy in the proposed framework is the exhaustive strategy, whose relevance is that it ensures obtaining the best configuration among all possible ones. In general, an exhaustive strategy is a brute-force method to a problem involving the search for a solution among all possible ones, e.g., those obtained from combinatorial objects, such as permutations or combinations. In this case, for a configurable process tree Q^{α}, we analyze all possible configurations that belong to $C(Q^{\alpha})$. Algorithm 1 presents the *exhaustive* strategy, which can be described as follows:

- Generate the set of all possible configurations, $C(Q^{\alpha})$, in a systematic manner, using the Cartesian product of the configuration sets for all configurable nodes.

- Loop over all configurations. At each iteration, the function *derive(*Q^{α}*,c)* is used to obtain a derived model m from the CPT Q^{α}, given the configuration *c*.

- The model *m* is evaluated using the *conformance* function defined in Equation 4.1.

- All potential configurations are evaluated, keeping track of the best solution found.

- After all configurations have been processed, the algorithm returns the configuration with the highest conformance.

Figure 5.1[1] presents an illustrative example of the exhaustive strategy. Given the CPT Q_1^α and the log L_1, shown in a) and b), the framework finds the best configuration and the corresponding derived process tree. The CPT Q_1^α, shown in a), has 4 configurable nodes, where the activity node *b* can be either *H* or *E*, the operator node $\wedge$ can be either *B* or *E*, the activity node *f* can be either *H* or *E*, and the activity node *g* can be either *H* or *E*. The set of configurations $C(Q^\alpha)$ contains $2^4 = 16$ different configurations. The framework checks every configuration $c \in C(Q^\alpha)$, shown in c). Among all possible feasible configurations, the algorithm selects c_{15} as the best configuration, shown in d). Once the best configuration is obtained, the framework applies it to the CPT Q_1^α to derive the process tree Q_{15}, shown in e).

[1]The algorithm generates all possible configurations, in order to find an optimal configuration. This configuration is used to obtain an optimal derived process tree.

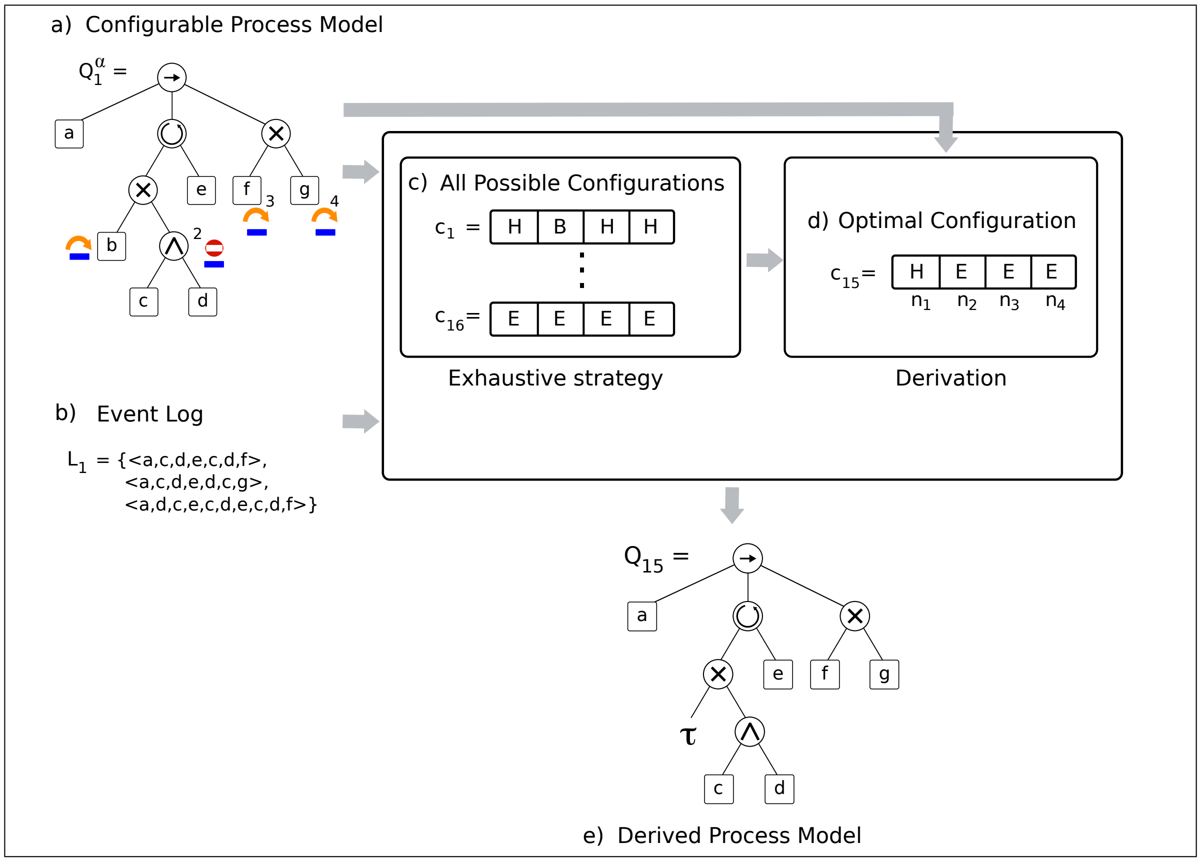


FIGURE 5.1. Overview of the Exhaustive strategy to derive a process tree.

Algorithm 1 Obtain a configuration based on the exhaustive strategy

```
1: procedure EXHAUSTIVE(LOG $L$, CPT $Q^\alpha$, WEIGHTS $W$)
2:     $best \leftarrow null$
3:     $C(Q^\alpha) \leftarrow CartesianProduct(\alpha(n_1), \alpha(n_2), \ldots, \alpha(n_{|N^\alpha(Q^\alpha)|}))$
4:     for all $c \in C(Q^\alpha)$ do
5:         $m \leftarrow derive(Q^\alpha, c)$
6:         if $conformance(m, L, W) > conformance(derive(Q^\alpha, best), L, W)$ then
7:             $best \leftarrow c$
8:     return $best$
```

5.2. Obtaining a configuration based on the genetic strategy

The exhaustive strategy requires a long time to find an optimal solution. Motivated to find a solution in less computing time, we have designed a second strategy to find a reasonable good configuration, based on a *genetic* evolutionary approach. *Genetic* algorithms (*GA*) are search algorithms that imitate the process of natural selection in nature, belonging to the class of evolutionary algorithms (Mitchell, 1998). They have successfully applied in the context of process mining for finding a process model that better represent the observed behavior in an event log (Buijs, van Dongen, & van der Aalst, 2012a; Lee, Choy, Ho, & Lam, 2016; Vázquez-Barreiros, Mucientes, & Lama, 2015). In this subsection, we present a *GA* approach to find a suitable configuration for a *CPT* model given a specific event log. The elements that define a *GA* are: representation of individuals, initialization, selection, crossover, mutation, and termination condition. Next, we present the setting of each of these elements in our configuration scenario. Figure 5.2[2] illustrates these main elements.

- **Representation :** In *GA*, a *chromosome* represents a potential solution and it is formed by a *genes* chain. Genes represent distinct aspects of the solution as a whole, just as human genes represent distinct aspects of people, such as their gender or eye color. A potential value of a gene is called an *allele*. In our case, a chromosome represents a configuration $c \in C(Q^{\alpha})$, see Figure 5.2 b), where each gene c_i corresponds to a configurable node n_i, and an allele is a configurator (B,H, or E) assigned to that particular configurable node, $c_i \in \alpha(n_i)$, for all $i \in 1, \ldots, |N^{\alpha}(Q^{\alpha})|$.

- **Initialization :** An initial population is generated randomly, where each individual represents a randomly created configuration c using valid *alleles*, i.e., $c_i \in \alpha(n_i)$. *Population size* is a parameter that determines the number of individuals in the first generation (Michalewicz, 1996).

[2]The internal configuration representation allows this evolutionary algorithm to use crossover and mutation operations to generate new candidate configurations to be assessed.

- **Selection :** In each generation, the best candidates are selected to move forward to the next generation, and some of them are also selected to be recombined. Each individual is evaluated using the conformance function[3] that evaluates the quality of a *chromosome* to either be selected for the next generation or to be discarded. We refer to the literature (Mitchell, 1998) to illustrate different selection strategies.

- **Crossover :** The crossover operation combines two parent chromosomes in order to generate two offspring chromosomes, as it is shown in Figure 5.2 c). Given two chromosomes a and b and a cutting point $1 \leq i < |N^{\alpha}(Q^{\alpha})|$, the offspring chromosomes are $\langle a_1, \ldots, a_i, b_{i+1}, \ldots, b_{|N^{\alpha}(Q^{\alpha})|}\rangle$ and $\langle b_1, \ldots, b_i, a_{i+1}, \ldots, a_{|N^{\alpha}(Q^{\alpha})|}\rangle$. Notice that, the proposed chromosome representation combined with the defined crossover operation produce only valid solutions, i.e., the offspring chromosomes are always valid configurations, according to the definitions presented in section 4.1.

- **Mutation :** A mutation produces a random change in one of the genes of the chromosome. In order not to produce spurious chromosomes, the mutation of a gene is restricted to the valid alleles of the gene, i.e., $c_i \in \alpha(n_i)$, where i is the mutated gene.

- **Termination conditions :** The most common alternatives for *GA* to terminate are: an upper limit for the number of generations, an upper limit for the conformance function (1 in our case), when the likelihood of achieving significant improvements in the next generation is very low, or when a given number of generation does not get any improvements (Mitchell, 1998).

[3]In GA theory, the function that evaluates the quality of a chromosome is called *fitness*. This fitness function does not correspond to the fitness function presented in Equation 4.1, but to the conformance function. Therefore and for the sake of clarity, we refer to it as conformance function.

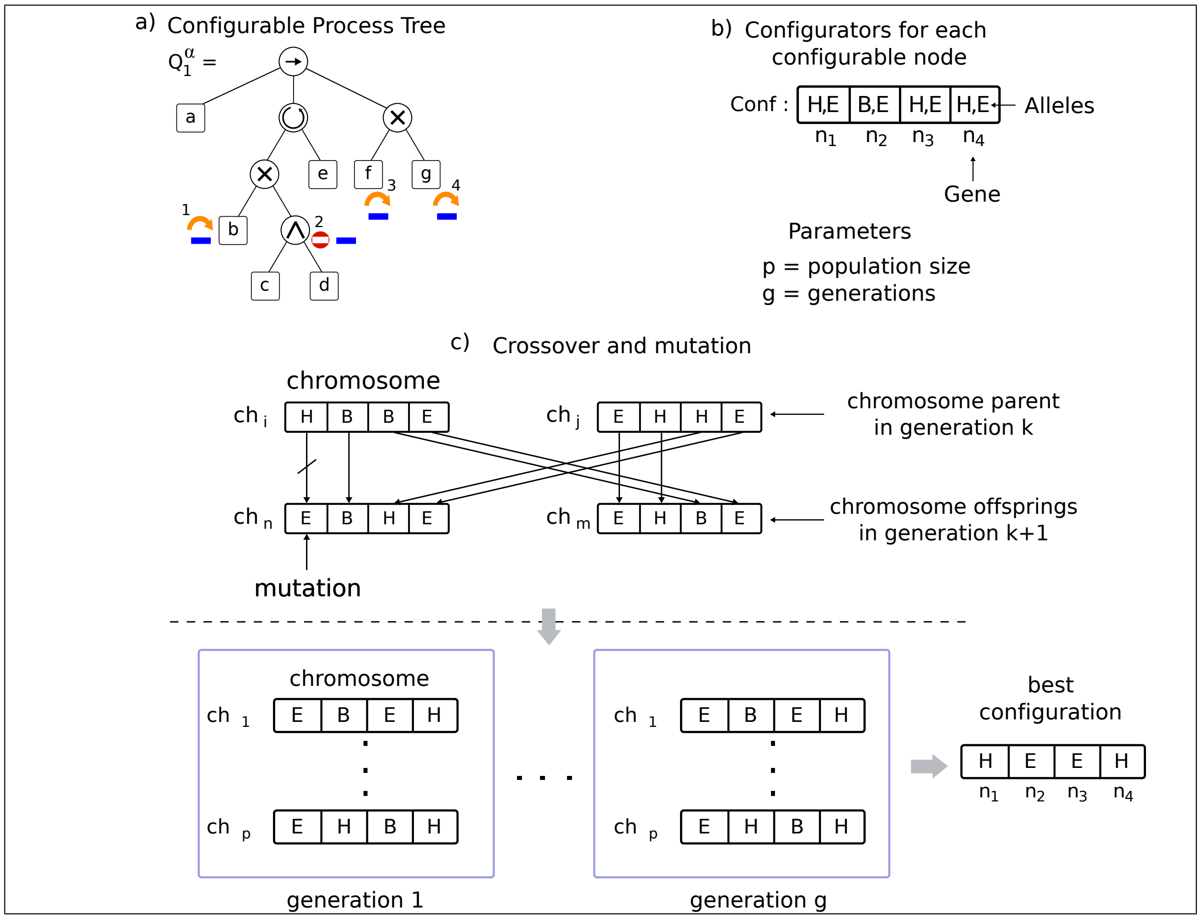


FIGURE 5.2. Overview of the Genetic strategy to derive a process tree.

Algorithm 2 describes the basic *GA* strategy. The main inputs are a configurable process model and an event log. The initialization of chromosomes is made in *initialPopulation()*, then all chromosomes of the initial population *pop* are evaluated using the *conformance* function in *bestIndividual(pop)*, in order to obtain the best individual. The population evolves over several generations until a termination condition is reached. In each generation, the algorithm selects qualified individuals through elitism in *selectParents(pop)*. Later, the function *crossover(parents)* recombines pairs of parents to create new individuals, in this way, a new population *pop'* is obtained. Mutation is then applied randomly to this new population, obtaining the new generation *pop"*. The best individual in the population *pop"* is then compared to the best configuration obtained so far. At the end, the algorithm returns the best individual (configuration) obtained using this evolutionary approach. For the sake of generality, Algorithm 2 describes the most generic *GA* strategy. More sophisticated techniques for each step of the algorithm are also possible (e.g., tournament, elitism, among others). Please refer to the literature (Buijs, 2014b; Mitchell, 1998) for more details.

Algorithm 2 Obtain a configuration based on the genetic strategy

```
1: procedure GENETIC(LOG $L$, CPT $Q^\alpha$, WEIGHTS $W$)
2:     $pop \leftarrow initialPopulation(Q^\alpha)$
3:     $best \leftarrow bestIndividual(pop)$
4:     while $not\ TerminationCondition()$ do
5:         $parents \leftarrow selectParents(pop)$
6:         $pop' \leftarrow crossover(parents)$
7:         $pop'' \leftarrow mutate(pop')$
8:         $m \leftarrow derive(Q^\alpha, bestIndividual(pop''))$
9:         if $conformance(m, L, W) > conformance(derive(Q^\alpha, best), L, W)$ then
10:            $best \leftarrow bestIndividual(pop'')$
11:    return $best$
```

5.3. Obtaining a configuration based on the greedy strategy

The above described evolutionary strategy is able to find a good configuration (potentially an optimal one (Mitchell, 1998)) in less time than the *exhaustive* strategy, but it is still time consuming. In order to reduce the time even more, but at the same time to be able to find a reasonably good configuration, we present a third strategy, based on a greedy heuristic. A *greedy* strategy is a heuristic search that creates a feasible solution incrementally, always making the choice that looks best at the moment of making a local choice. Sometimes these local choices lead to a global optimal solution (Cormen, Leiserson, Rivest, & Stein, 2009). Depending on the problem and search space, this strategy does not result in finding one of the optimal solutions, however for many problems they provide a close to optimal solution (even an optimal one) in a reasonable computing time.

Greedy algorithms usually divide the problem in small sub-problems; each sub-problem is then solved independently and in an incremental fashion. In our case, we can take sub-trees from the configurable process tree and process each of them independently, and make *good local choices* in the hope that they result in *an optimal solution* when we apply all these local configuration choices to the configurable process model.

In a configurable process tree, two (or more) configurable nodes are *dependent* if one of them is under or above the other one in a tree branch or if they both have a common ancestor that is a ↺ operator. If so, the configuration of one of these nodes might affect the configuration of the other one. On the other hand, a configurable node is *independent* if it is not *dependent* of any other configurable node.

We can identify three scenarios in a configurable process tree depending on the dependency among its configurable nodes. The configurable process tree can have only *independent* configurable nodes, as shown in Figure 5.3 (a); it can have only *dependent* configurable nodes, as shown in Figure 5.3 (b); or it can combine both *independent* and *dependent* configurable nodes, as shown in Figure 5.3 (c).

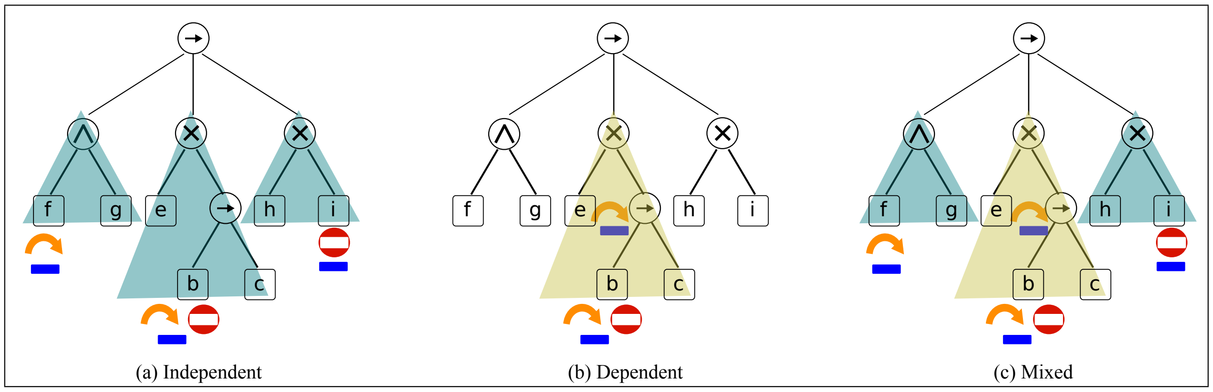


FIGURE 5.3. Different configurable node dependencies.

Algorithm 3 describes the proposed greedy strategy, which consists of the following steps:

- The configurable process tree is traversed to obtain a sorted list of all configurable nodes. A hierarchical order is achieved by applying the following rules:

  - *Dependent* configurable nodes have a higher priority than *independent* configurable nodes.

  - Among configurable nodes of the same type (*dependent* or *independent* configurable nodes), a deeper configurable node has a higher priority.

  - Among configurable nodes in the same level, an operator node has a higher priority than an activity node; otherwise, they are sorted from left to right.

- Every configurable node is then processed accordingly to its priority. For each configurable node, a subtree and a sublog are obtained to compute the local conformance:

  - To obtain a subtree for a configurable node, a new root has to be considered. If a configurable node is an activity node, the new root is its direct parent, so that a subtree always has an operator as a root; otherwise, if it is an operator node, the new root is the own operator node.
    If the configurable node has some ancestors that are loop operators, then the new root is the loop operator ancestor that is closer to the original root. Such a new subtree might contain other pending configurable nodes; they are temporarily set to τ in order to postpone any decision about their configuration.

  - To obtain a sublog, we get the projection of the event log on the set of activities contained in the subtree.

- For the selected configurable node, all possible configurators are evaluated, obtaining different derived process subtrees. The local conformance between each of those process subtrees and the event sublog is computed. The best configurator is saved and then set in the best configuration for the original configurable process tree.
- At the end, the best configuration for the whole configurable process tree is returned.

Figure 5.4[4] illustrates the greedy strategy to find a configuration for the configurable process tree Q_1^{α}. The derivation process is shown on the bottom part of the figure for two logs, L_1 and L_2, where, for every configurable node, the best configurator is selected among all feasible configurations for each configurable node. First, the order in which the configurable nodes will be processed is decided. n_1 and n_2 are *dependent* nodes because they have a common ancestor that is a $\circlearrowleft$ operator. Meanwhile, n_3 and n_4 are *independent* nodes. Hence, n_1 and n_2 have a higher priority than n_3 and n_4. Since n_2 is an operator node, it has a higher priority than n_1. n_3 and n_4 are both activity nodes, so they are prioritized from left (n_3) to right (n_4). Therefore, the order is n_2, n_1, n_3, n_4, regardless of the event log that will be considered. For the event log L_1, the algorithm starts from the deeper node n_2. n_1 is set to τ and all possible configurators (*B* and *E*) for n_2 are then evaluated, considering the subtree that has as a root the loop operator that is an ancestor of n_2, and the sublog obtained projecting the original log L_1 on the activities contained in the subtree: c, d, e. The best configuration for n_2 is *E*. Later, having the configuration of n_2, in a similar way n_1 is analyzed and configured to *E*. Afterwards, n_4 is set to τ while n_3 is configured to *E*. Finally, once n_3 is already configured, the last node n_4 is configured to *H*. The final configuration for L_1 is then obtained, and represented as the *best configuration*. For the event log L_2, the algorithm proceeds in a similar way. Notice that since the log L_2 contains fewer activities that the configurable process tree Q_1^{α}, the projection of the activities contained

[4]There are two event log inputs to show different subtrees and sublogs scenarios to finally derive a process tree.

in the subtrees creates very simple sublogs, even an empty event log, such as the obtained when processing the configurable node n_2. The best configuration in this case considers to block the configurable node n_2 and hide the configurable node n_4, illustrating how the algorithm adapts to different scenarios. As a result, Q_1 is the derived process tree from Q_1^α and L_1, and Q_2 is the derived process tree from Q_1^α and L_2.

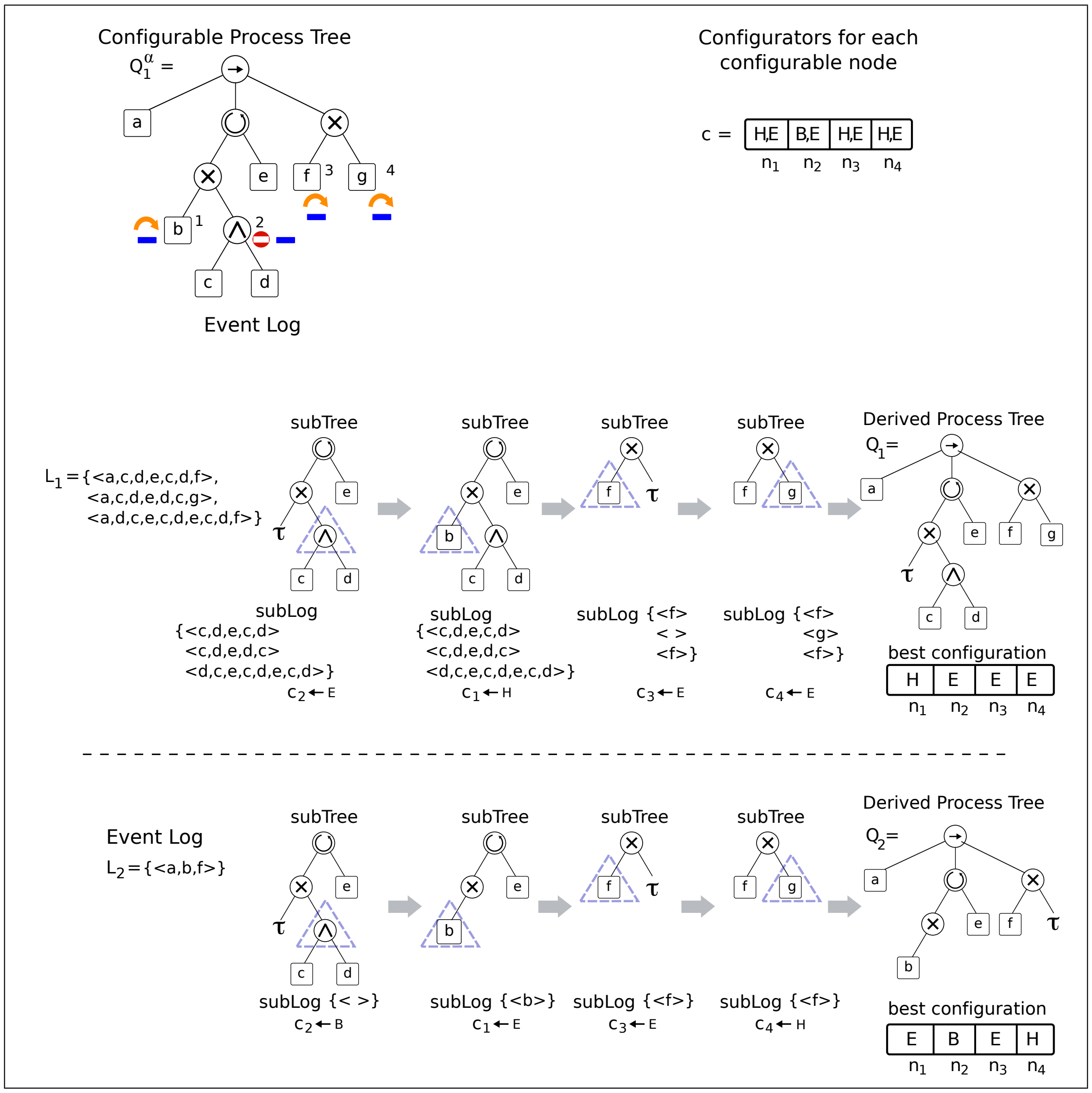


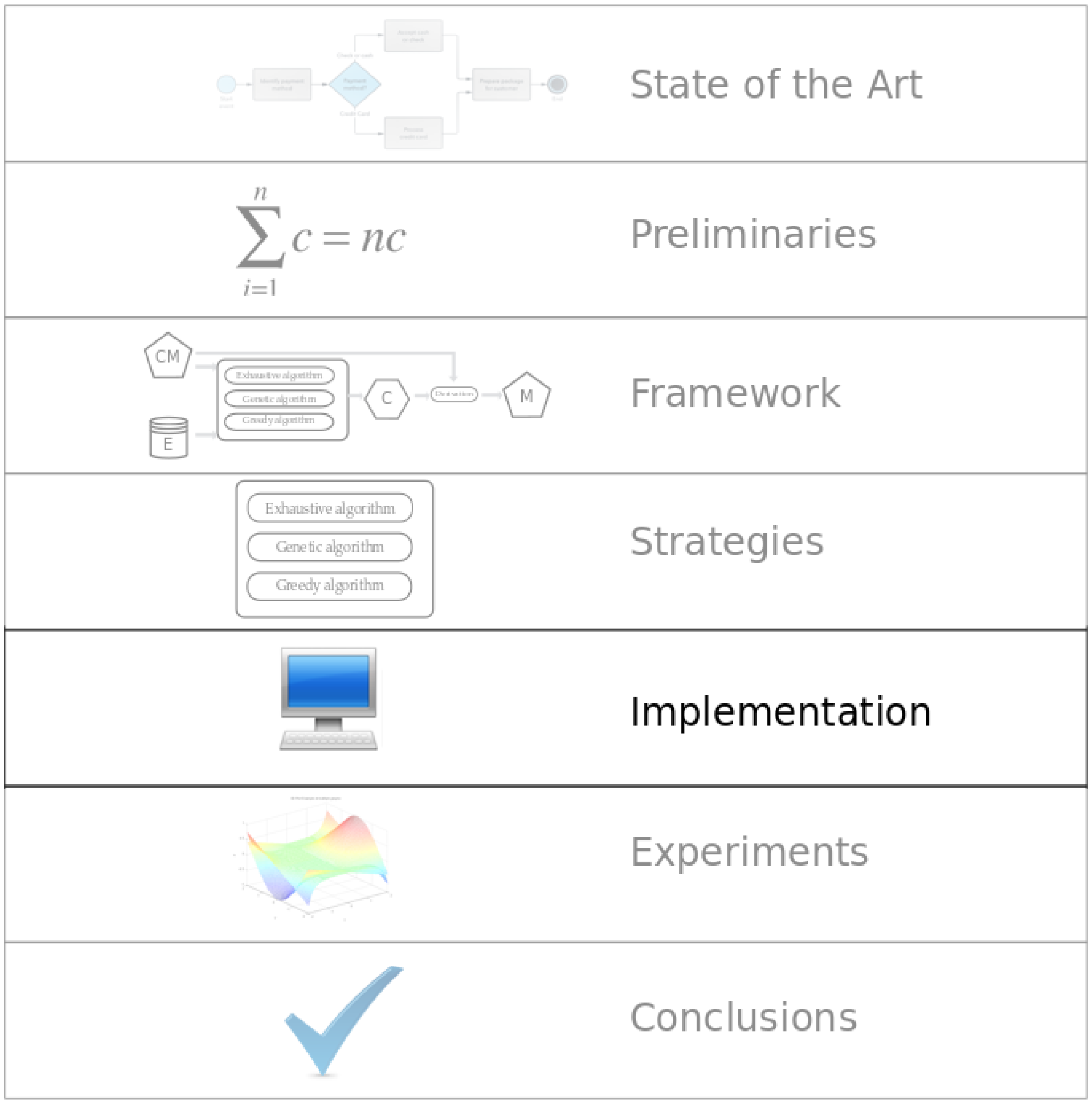
FIGURE 5.4. Overview of the Greedy strategy to derive a process tree.

Algorithm 3 Obtaining a configuration based on the greedy strategy

```
1: procedure GREEDY(LOG $L$, CPT $Q^\alpha$, WEIGHTS $W$)
2:     $configurableNodeList \leftarrow getPrioritizedConfigurableNodes(Q^\alpha)$
3:     $bestConfiguration \leftarrow emptyConfiguration()$
4:     for all $cn \in configurableNodeList$ do
5:         $sQ^\alpha \leftarrow getSubTreeFromConfigurableNode(Q^\alpha, cn)$
6:         $activityList \leftarrow getActivityNodes(sQ^\alpha)$
7:         $sL \leftarrow L\upharpoonleft_{activityList}$
8:         $best \leftarrow null$
9:         for all $configurator \in \alpha(cn)$ do
10:            $m \leftarrow derive(sQ^\alpha, configurator)$
11:            if $conformance(m, sL, W) > conformance(derive(sQ^\alpha, best), sL, W)$ then
12:                $best \leftarrow configurator$
13:        $bestConfiguration_{cn} \leftarrow best$
14:    return $bestConfiguration$
15:
16: procedure $getSubTreeFromConfigurableNode$(CPT $Q^\alpha$, CONFIGURABLENODE $cn$)
17:    if $IsActivityNode(Q^\alpha, cn)$ then
18:        $nodeRoot \leftarrow getParent(Q^\alpha, cn)$
19:    else if $IsOperatorNode(Q^\alpha, cn)$ then
20:        $nodeRoot \leftarrow cn$
21:    for all $n \in Ancestors(Q^\alpha, nodeRoot)$ do
22:        if $IsLoopOperatorNode(Q^\alpha, n)$ then
23:            $nodeRoot \leftarrow n$
24:    $sQ^\alpha \leftarrow getSubTree(Q^\alpha, nodeRoot)$
25:    for all $n \in Descendants(sQ^\alpha, nodeRoot)$ do
26:        if $n \neq cn$ && $IsConfigurableNode(sQ^\alpha, n)$ then
27:            $n \leftarrow \tau$
28:    return $sQ^\alpha$
```

6. IMPLEMENTATION

In this chapter we present the implementation of the exhaustive, genetic and greedy strategies in the open-source framework for process mining ProM (Verbeek, Buijs, van Dongen, & van der Aalst, 2010), (van der Aalst et al., 2009).



6.1. Creation of the configurable process model

We have implemented the three strategies: exhaustive, genetic, and greedy, as three plug-ins of the ProM process mining framework (van der Aalst et al., 2009), within the *ConfigurableProcesses* package[1]. The genetic evolutionary strategy is implemented using *JGAP*, a Java library for *GA*; this flexible library fits in our genetic evolutionary approach. All experiments have been performed in a laptop with an Intel Core i5 CPU at 2,7 GHz, 8GB RAM, running OS X El Capitan 64 bits.

The process model used in this research is based on a process tree representation. Thus, the *ConfigurableProcesses* package uses the *ProcessTree* package[2] to represent our model. As presented in Figure 1.2, the framework requires to use the step **Derivation Strategies** to obtain a configuration. In order to evaluate the quality of the derived process tree based on the obtained configuration, each strategy applies conformance checking on the derived process tree. Conformance checking is implemented in the *ProcessTreeReplay* package[3], also included in our package.

FIGURE 6.1. Overview of the ProM framework.

[1]Available in the ProM nightly-builds `http://www.promtools.org/prom6/nightly`
[2]`https://svn.win.tue.nl/trac/prom/browser/Packages/ProcessTree/Trunk`
[3]`https://svn.win.tue.nl/trac/prom/browser/Packages/ProcessTreeReplayer/Trunk`

The main interface of ProM framework is shown in Figure 6.1, where it is possible to upload files such as a CPT and an event log, using the **import** button. Once the files are uploaded, the main tab of the framework shows the uploaded elements that are ready to use by pressing the **play** button (play icon), as depicted in Figure 6.2.

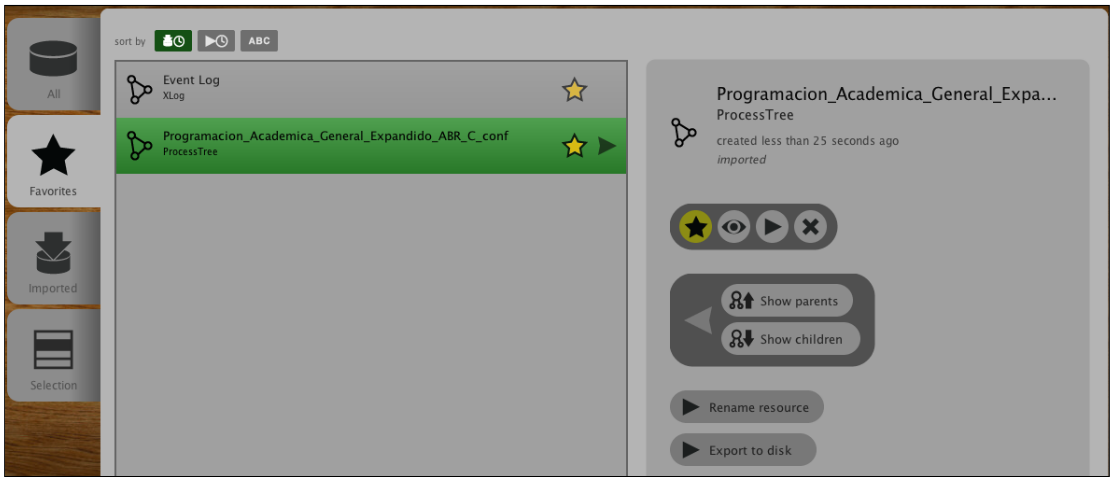
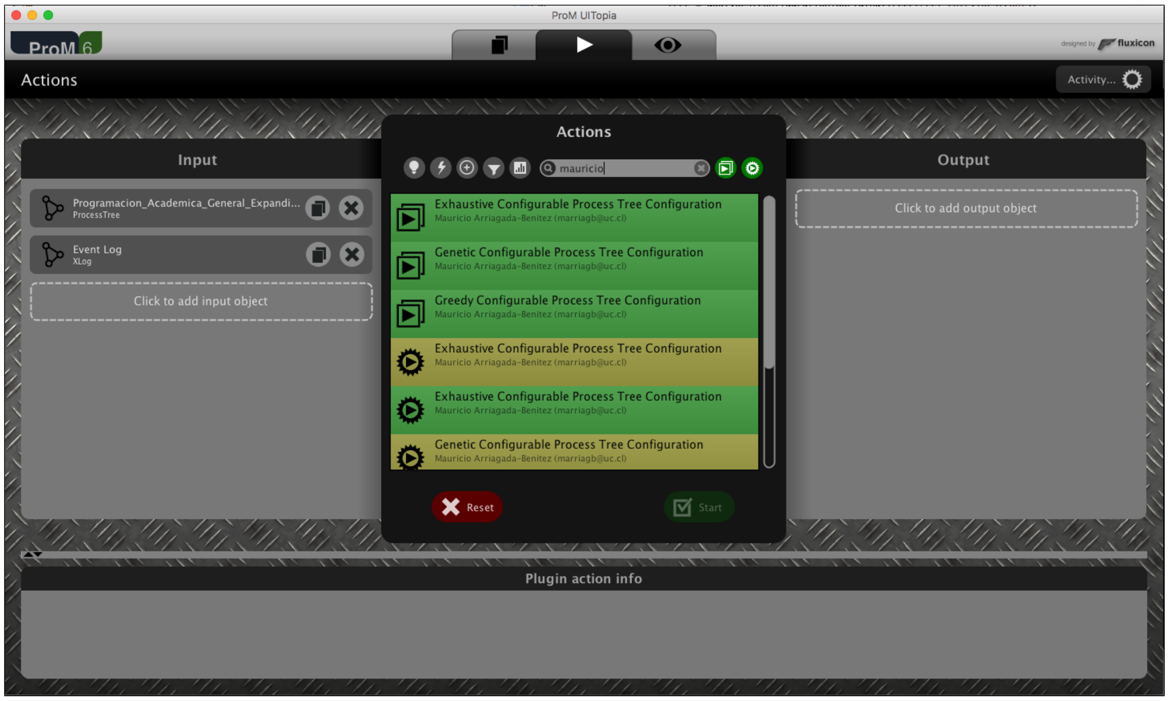


FIGURE 6.2. Overview of the inputs of the ProM framework.

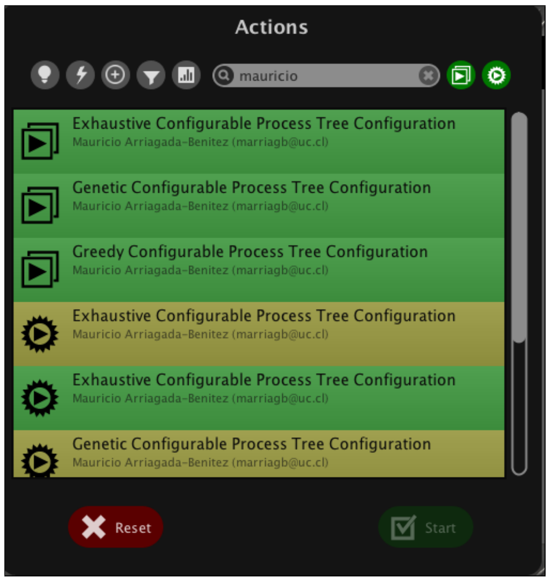
After pressing the **play** button, a new window in the ProM framework will display the different algorithms available to use, as shown in Figure 6.3. Each algorithm guides the user on how to use such algorithm giving information of the different input needed and also what is the expected output. Thus, after loading a CPT and an event log, our strategies appear in green color as available plug-ins to use.



Exhaustive Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Genetic Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Greedy Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Exhaustive Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Exhaustive Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Genetic Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)

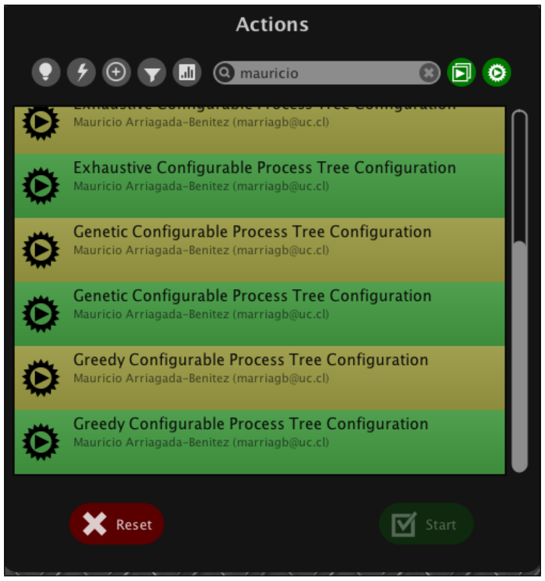
FIGURE 6.3. Overview of the strategies we created as plug-ins in the ProM framework.

It is possible to run the strategies using a user interface, as shown in Figure 6.4, or without using a parameter setting file where values are specified for all parameters, as shown in Figure 6.5. The main difference between both ways to run the strategies is that the plug-in without user interface facilitates automatic testing.



Exhaustive Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Genetic Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Greedy Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Exhaustive Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Exhaustive Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Genetic Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)

FIGURE 6.4. Overview of the strategies we created as plug-ins in the ProM framework. This figure shows the plug-ins that have a user interface.



Mauricio Arriagada-Benitez (marriagb@uc.cl)
Exhaustive Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Genetic Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Genetic Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Greedy Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)
Greedy Configurable Process Tree Configuration
Mauricio Arriagada-Benitez (marriagb@uc.cl)

FIGURE 6.5. Overview of the strategies we created as plug-ins in the ProM framework. This figure shows the plug-ins that do not have a user interface.

6.2. Running the plug-ins

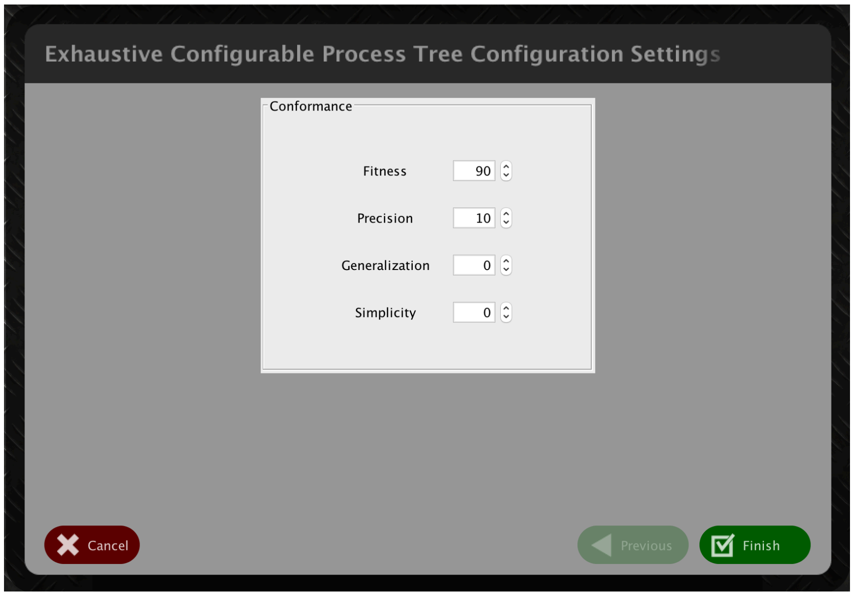
The *ConfigurableProcesses* package has been tested using the running example of the configurable process model presented on chapter 3 (Figure 4.1). The synthetic log was created using the *Log Generator* plug-in implemented by vanden Broucke, De Weerdt, Baesens, and Vanthienen (2012) that uses our process model, in a Petri Net representation, as input. The summary of the artificial log is shown in Table 6.1.

TABLE 6.1. Synthetic log of the running example.

trace	#traces
ABG	7
ABF	7
ADCG	7
ABEBF	4
ACDF	3
ADCF	3
ACDEDCG	1
ACDG	2
ADCEBF	2
ACDEBEDCF	1
ACDEDCF	1
ACDEBEBG	1
ABEDCF	1
ACDECDF	1
ABEDCEBG	1
ACDEDCEBF	1
ACDEBG	1
ABEDCG	1
ABECDEBG	1
ABECDG	1
ACDECDG	1
ACDEBF	1

6.2.1. Running the exhaustive strategy

Given a configurable process tree and a log, the exhaustive plugin explores exhaustively all possible configurations and selects the most conformant one. Once the two inputs are entered, the plugin pops up a windows with the conformace parameters to be set. By default, we considered 90% of fitness and 10% of precision, as shown in Figure 6.6. These values follow the recommendations made by (Buijs et al., 2012b), in which it was calculated using Pareto front that these values allow to obtain a good balance between fitness and precision.


Exhaustive Configurable Process Tree Configuration Settings

Conformance

Fitness 90

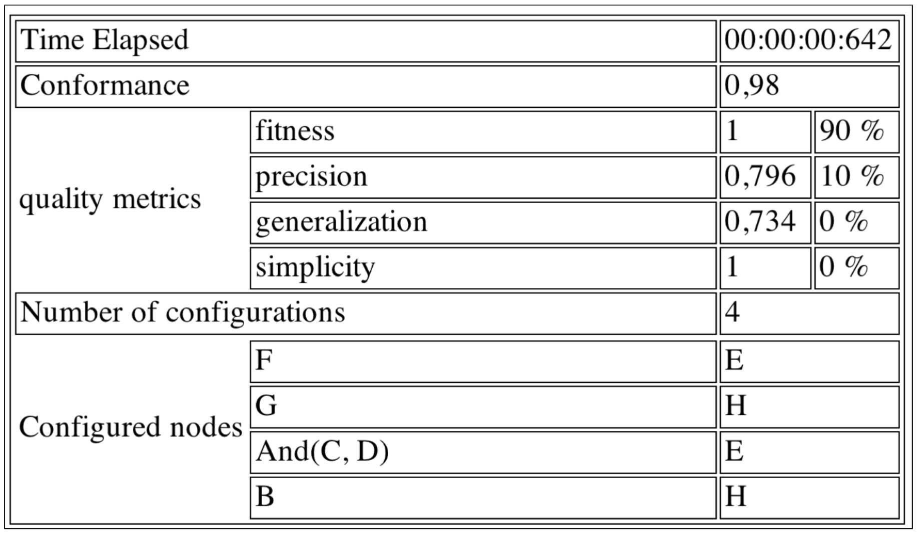
Precision 10

Generalization 0

Simplicity 0

FIGURE 6.6. Exhaustive strategy setting.

The outputs of this plug-in are both a summary of the configuration and a configured process model, as shown in Figure 6.7 and Figure 6.8, respectively.



Time Elapsed		00:00:00:642	
Conformance		0,98	
quality metrics	fitness	1	90 %
	precision	0,796	10 %
	generalization	0,734	0 %
	simplicity	1	0 %
Number of configurations		4	
Configured nodes	F	E	
	G	H	
	And(C, D)	E	
	B	H	

FIGURE 6.7. Summary obtained by the exhaustive strategy plug-in.

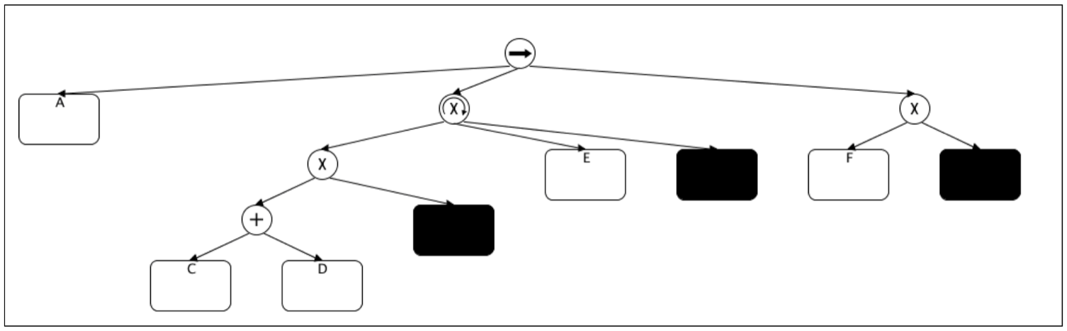
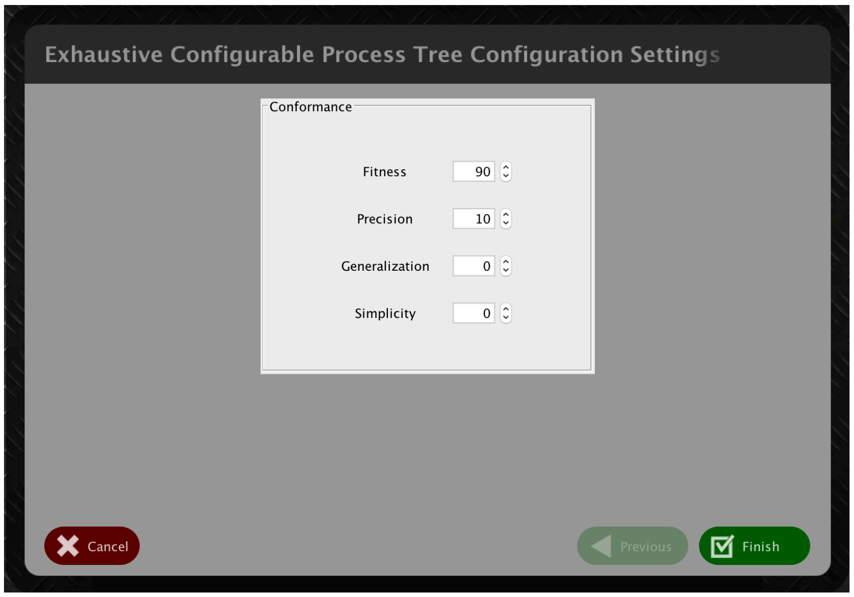


FIGURE 6.8. Derived process tree obtained by the exhaustive strategy plug-in.

Figure 6.7 depicts that the exhaustive strategy took 642 milliseconds to derive a process tree with a conformance of 0.98. It also shows the four quality metrics that the derived process tree has: fitness 1, precision 0.796, generalization 0.734, and simplicity 1. In addition, this summary gives the number of configurations that the configurable process model has. In this case, it has 4 configurations. Finally, it shows each configurable node and its found configuration. In this example, node **F** is *enabled*, node **G** is *hidden*, node **And(C,D)** is *enabled*, and node **B** is *hidden*, as expected. The derived process tree is shown in 6.8.

6.2.2. Running the genetic strategy

Given a configurable process tree and a log, the genetic plug-in explores the configurations and selects the most conformant one. Once the two inputs are entered, the plug-in pops up a windows with the conformace parameters to be set. By default, we considered 90% of fitness and 10% of precision. Genetic strategy also allows to set the number of generation and population. We have set by default 20 generations and a population of 10 individuals, as shows in Figure 6.9.


Exhaustive Configurable Process Tree Configuration Settings

Conformance

Fitness 90

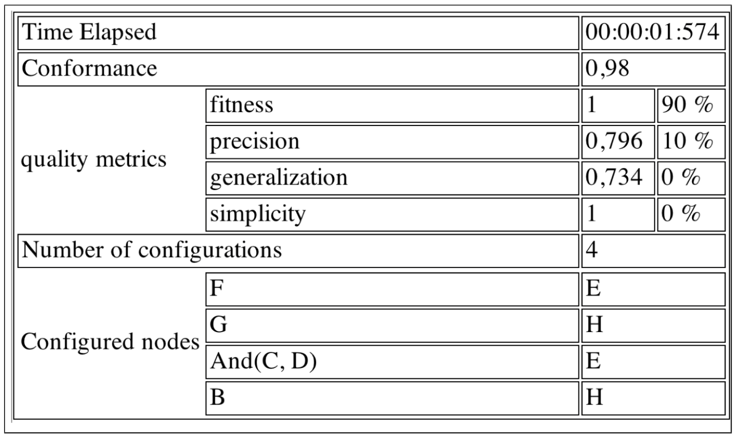
Precision 10

Generalization 0

Simplicity 0

FIGURE 6.9. Genetic strategy setting.

The outputs of this plug-in are both a summary of the configuration and a configured process model, as shown in Figure 6.10 and Figure 6.11, respectively.



Time Elapsed		00:00:01:574	
Conformance		0,98	
quality metrics	fitness	1	90 %
	precision	0,796	10 %
	generalization	0,734	0 %
	simplicity	1	0 %
Number of configurations		4	
Configured nodes	F	E	
	G	H	
	And(C, D)	E	
	B	H	

FIGURE 6.10. Summary obtained by the genetic strategy plug-in.

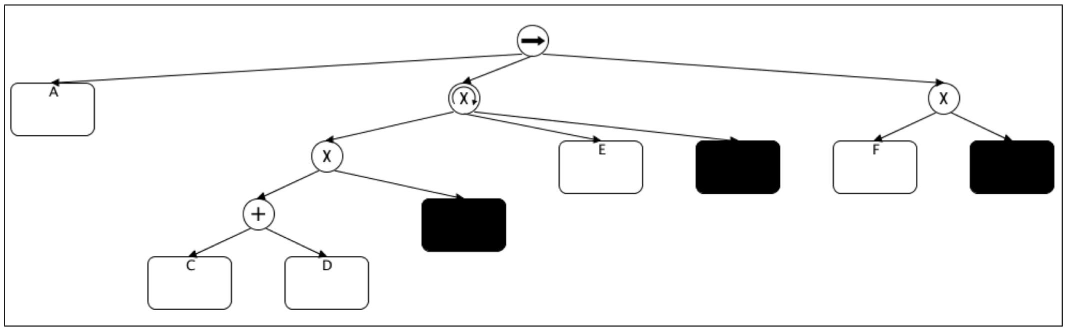
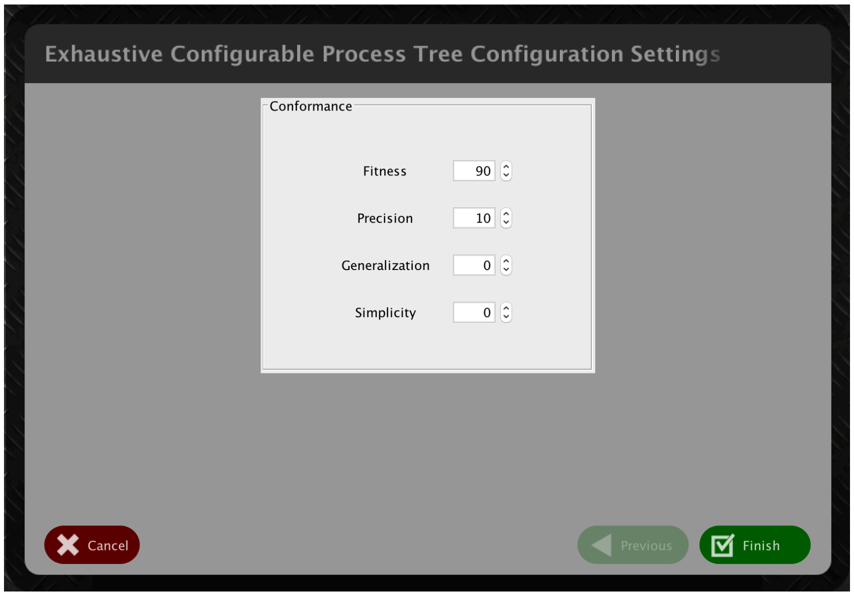


FIGURE 6.11. Derived process tree obtained by the genetic strategy plug-in.

Figure 6.10 depicts that the genetic strategy took 1 second and 574 milliseconds to derive a process trees with a conformance of 0,98. It also shows the four quality metrics that that derived process tree has: fitness 1, precision 0.796, generalization 0.734, and simplicity 1. In addition, this summary gives the number of configurations that the configurable process model has. In this case it has 4 configurations. Finally, this summary shows each configurable node and its found configuration. In this example, node **F** is *enabled*, node **G** is *hidden*, node **And(C,D)** is *enabled*, and node **B** is *hidden*, as expected. The same result was found with the previous strategy. Since this is a small example—a small configurable process model with a few configurable nodes—this strategy takes more time than the exhaustive strategy due to the large number of configuration to check due to the others two parameters that this strategy considers: population and generations. When a configurable process model has large configurable nodes to configure, this strategy takes less time than the exhaustive strategy. The derived process tree is shown in 6.11.

6.2.3. Running the Greedy strategy

Given a configurable process tree and a log, the greedy plug-in explores the configurations and selects the most conformant one in a short time. Once the two inputs are entered, the plug-in pops up a windows with the conformace parameters to be set. By default, we considered 90% of fitness and 10% of precision, as shows in Figure 6.12.

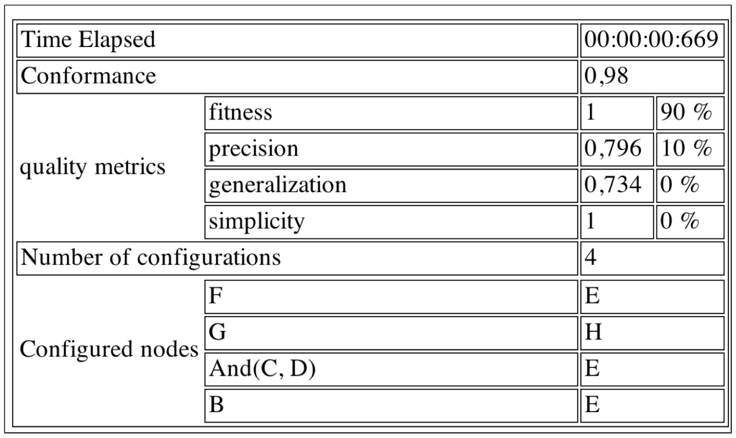

Exhaustive Configurable Process Tree Configuration Settings

Conformance

Fitness	90
Precision	10
Generalization	0
Simplicity	0

FIGURE 6.12. Greedy strategy setting

The outputs of this plug-in are both a summary of the configuration and a configured process model, as shown in Figure 6.13 and Figure 6.14, respectively.



Time Elapsed		00:00:00:669	
Conformance		0,98	
quality metrics	fitness	1	90 %
	precision	0,796	10 %
	generalization	0,734	0 %
	simplicity	1	0 %
Number of configurations		4	
Configured nodes	F	E	
	G	H	
	And(C, D)	E	
	B	E	

FIGURE 6.13. Summary obtained by the greedy strategy plug-in.

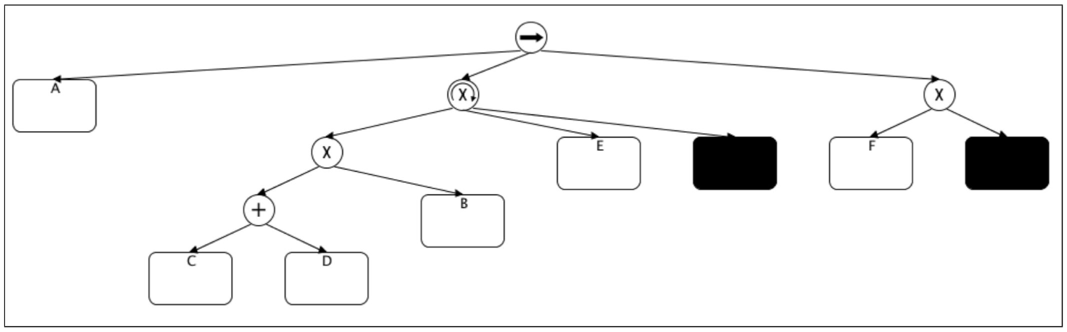


FIGURE 6.14. Derived process tree obtained by the greedy strategy plug-in.

Figure 6.13 depicts that the genetic strategy took 669 milliseconds to derive a process trees with a conformance of 0.98. It also shows the four quality metrics that the derived process tree has: fitness 1, precision 0.796, generalization 0.734, and simplicity 1. In addition, this summary gives the number of configurations that the configurable process model has. In this case it has 4 configurations. Finally, this summary shows each configurable node and its found configuration. In this example, node **F** is *enabled*, node **G** is *hidden*, node **And(C,D)** is *enabled*, and node **B** is *enabled*. We were expecting to have the node **B** as *hidden*. This strategy finds the best local solution for every single node, but it might not represent the best global solution. When a configurable process model has large configurable nodes to configure, this strategy takes a considerable less time than the exhaustive and genetic strategies. The derived process tree is shown in 6.14.

7. EXPERIMENTS AND DISCUSSIONS

In this chapter we describe the two experimental scenarios considered: a realistic scenario based on the adoption of a higher educational ERP system by some universities (see section 7.1), and a scenario that represents a real-life registration process (Gottschalk et al., 2009), which is executed on a daily basis by Dutch municipalities (see section 7.2). Afterwards, in section 7.3, we analyze the algorithm's performance based on an empirical evaluation.

7.1. Educational scenario

Educational institutions such as universities are spread along cities in a country with the purpose of granting academic degrees in various subjects. When the owner of a network of universities decides to standardize the higher educational ERP system to be used for all the universities belonging to the network, the first step is to know how suitable the software is for each university.

The ERP system provides support for different processes, and it can be configured to suit how each process is executed in the university where it will be installed. The diversity of process variants the ERP supports can be represented through a configurable process model. The decisions that are made to adapt the software to the way the process is executed in each university corresponds to the configuration that allows to generate a derived process model specific for each university. The derived process model will probably not be exactly the same as the process model that represents how the process is currently executed, but it will be the closest process variant the software is able to support.

If a university is currently running the process using other software, we can assume an event log is currently recording how each process is being executed.

The framework proposed in this thesis takes as input the configurable process model that represents the different parameterizations provided by the ERP system for the process and the event log that represents the current execution of the process in a university. Based on them, the framework generates a derived process model that represents how the process could be run in the future using the ERP system.

7.1.1. Academic planning process

Beyond their organizational structure, universities share some common processes, such as planning academic courses. The planning of academic courses in a university is a complex process due to several factors such as government regulations, internal policies, dynamic changes of knowledge in certain domains, economic and human resources, among others. In this process, both administrative personnel and faculty members, sometimes from different departments, are involved in each stage of the process. This process, in general, considers some similar stages across universities such as course planning, student assistant planning, thesis planning, among others. To deal with its complexity, it can be modeled at a high-level. Figure 7.1 depicts a general process model that includes sub-processes that group common activities in the academic planning process. We refer to appendix A to observe details of the subprocesses of the Figure 7.1[1].

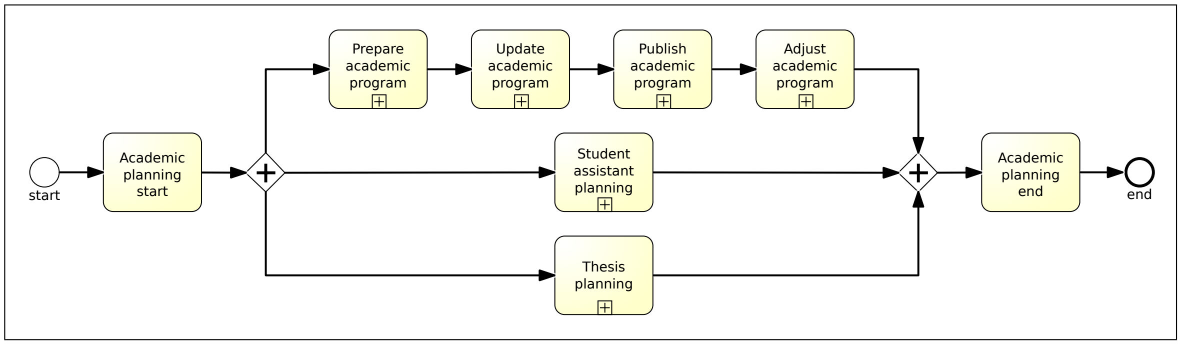


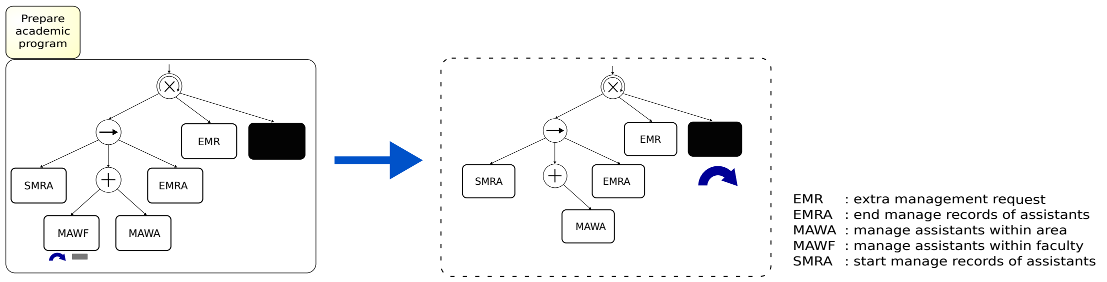
FIGURE 7.1. General model of the academic planning process of a university.

We have used this process model as a reference process model for three different universities located along a country. For the sake of simplicity, we call them: *Northern* University, *Central* University, and *Southern* University, according to their location in the country. Each university executes a different process variant of the reference model according to its policies, regulations, socio-demographic characteristics, and geographic needs. For example, the *Northern* University is a low-income university, so there are some courses that do not have student assistants, whereas in the other two high-income universities there is more than one student assistant in some courses. The *Southern* University does not have a proper internal information system, so selecting student assistants is a manual task.

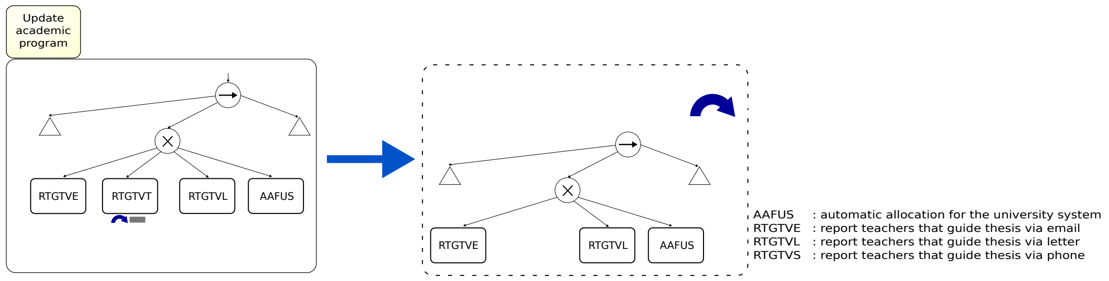
The configurable process tree contains 70 activities, which can be simplified into 6 sub-processes and 2 activities shown in Figure 7.1, and 9 configurable nodes that have *mixed* dependencies. The configurable process model is considerably large, making it impractical

[1]It includes parallel flows that contain sub-processes.

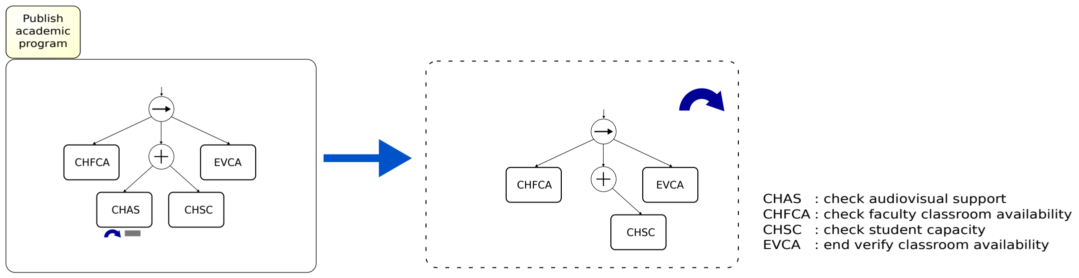
to show it completely. To illustrate how the configuration is performed, Figure 7.2 shows a scheme of the configuration process. The upper boxes represent partial configurable process trees for three different sub-processes of the process reference model. The lower process trees, inside the dash line boxes, are derived process trees where the corresponding configurable nodes have been configured as hide.


Prepare academic program
EMR
SMRA
EMRA
MAWF
MAWA
SMRA
EMRA
EMR
MAWA
EMR : extra management request
EMRA : end manage records of assistants
MAWA : manage assistants within area
MAWF : manage assistants within faculty
SMRA : start manage records of assistants

(A) Northern University.


Update academic program
RTGTVE
RTGTVT
RTGTVL
AAFUS
RTGTVE
RTGTVL
AAFUS
AAFUS : automatic allocation for the university system
RTGTVE : report teachers that guide thesis via email
RTGTVL : report teachers that guide thesis via letter
RTGTVS : report teachers that guide thesis via phone

(B) Central University.


Publish academic program
CHFCA
EVCA
CHAS
CHSC
CHFCA
EVCA
CHSC
CHAS : check audiovisual support
CHFCA : check faculty classroom availability
CHSC : check student capacity
EVCA : end verify classroom availability

(C) Southern University.

FIGURE 7.2. General idea of process model derivation for three different configurable sub-processes.

Having a reference process model, whose overall view is shown in Figure 7.1, and an academic planning event log per university, it was possible to derive a process model for each university. We applied the proposed *exhaustive*, *genetic*, and *greedy* strategies to the three universities. The results are shown in Table 7.1, including fitness, precision, and the conformance metric, calculated as 90% fitness and 10% precision.

It is possible to observe in Table 7.1 that both *Northern* University and *Central* University have a high conformance, and also a high fitness. We can assert that both universities have a high percentage of commonality with the configurable process model. However, that is not the case for *Southern* University, which has a lower conformance and a lower fitness. One of the reasons for this is the considerable amount of activities that are being executed at *Southern* University that are not found in the reference model. A decision maker would probably decide the ERP system that is desired to acquire is not suitable for this university or that the process must be completely changed in *Southern* University.

TABLE 7.1. Comparison of the proposed strategies for the three universities.

	Northern			Central			Southern		
	fitness	precision	conformance	fitness	precision	conformance	fitness	precision	conformance
Exhaustive	0.981	0.398	**0.922**	0.986	0.510	**0.939**	0.690	0.510	**0.672**
Genetic	0.981	0.398	**0.922**	0.986	0.510	**0.939**	0.690	0.510	**0.672**
Greedy	0.981	0.370	**0.920**	0.986	0.500	**0.938**	0.690	0.500	**0.671**

Table 7.1 shows that all three strategies obtained process trees with the same fitness. However, the greedy strategy obtained a process tree with a lower precision. Taking the Northern University as an example, a qualitative analysis between the process tree obtained by the exhaustive strategy and the greedy strategy suggests that the exhaustive strategy finds a configuration of the model where the activity Assign Classroom Automatically (ACA) is always performed (as observed in the event log), while the greedy strategy finds a configuration of the model where the activity ACA can either be performed or skipped (that was never observed in the event log), obtaining a lower precision.

The performance of the three strategies is depicted in Figure 7.3. It is possible to observe that the time required by the *exhaustive* strategy was considerably higher than the time required by the *genetic* and *greedy* strategies. Moreover, the *greedy* approach was able to obtain a derived process model in seconds whereas the other two algorithms required minutes and even hours. The time required in the case of the *Southern* University is higher due to the alignment technique used to obtain the conformance metric. As it was mentioned before, at the *Southern* University, many activities are being executed that are not found in the reference model; in those cases, the alignment technique takes longer (Adriansyah, 2014).

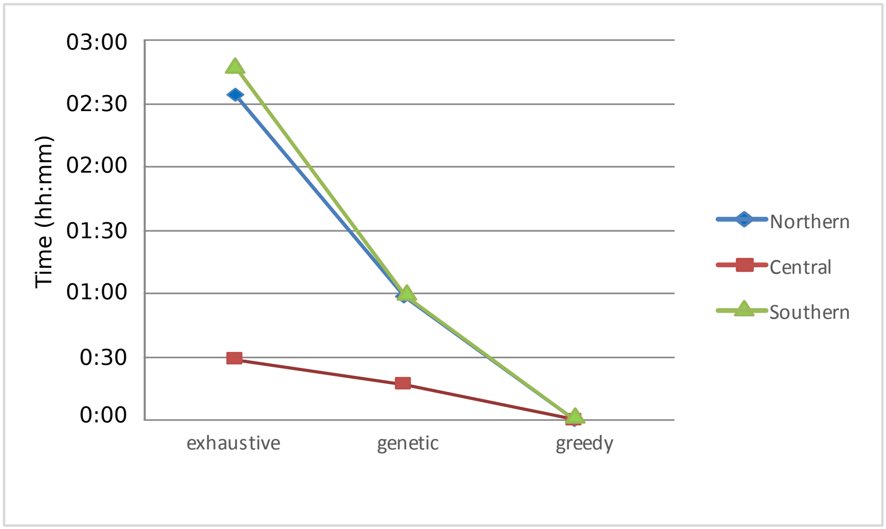


FIGURE 7.3. Benchmarking of the performance of the three strategies.

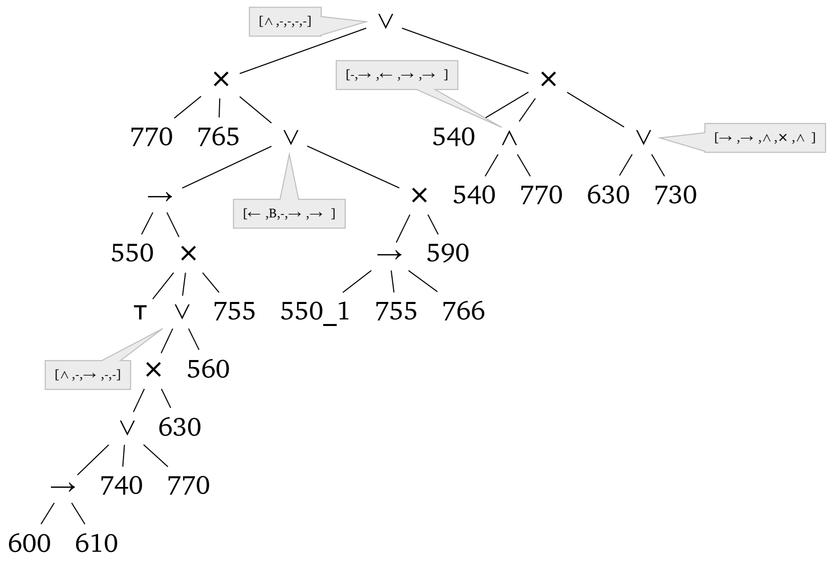
In conclusion, the *greedy* strategy is able to obtain in a short time a derived process model that has a quite good conformance, in comparison to the conformance obtained with the other two strategies.

7.2. Municipal scenario

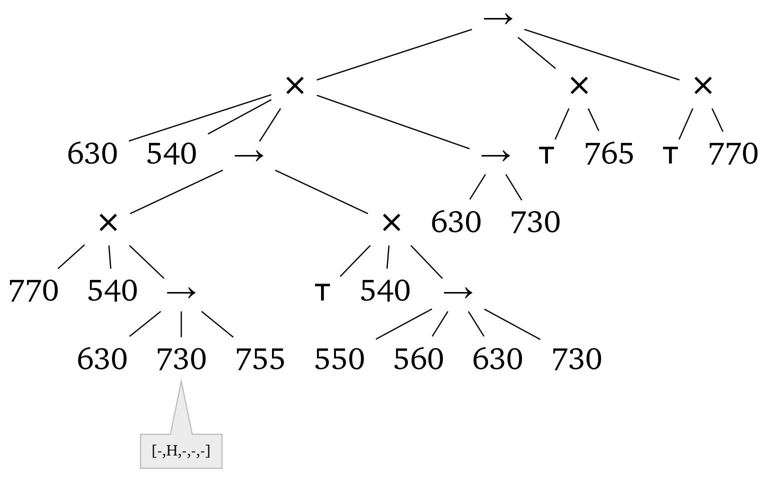
In order to evaluate the performance of the techniques on real-life data, we applied all three strategies on a real-life dataset. The data used contains event data from the building permits process of five Dutch municipalities (Van Dongen, 2015; Buijs, 2014a). There are five event logs, each describing a different process variant, which were extracted from the IT systems of the corresponding municipality. The same data has been used to evaluate the performance of configurable process discovery of the Evolutionary Tree Miner (ETM) (Buijs, 2014b, pages 250 and 251). In this experiment, we create some CPTs based on two of the models discovered by the ETM on this dataset. We then allow each of the three strategies to configure the CPTs to come to the best solution they can achieve. We then compare these results with the results obtained by the ETM on the same dataset.

7.2.1. Real-life experiments

We use CPTs based on the models discovered by the ETM on this dataset, which are shown in Figure 7.4. Figure 7.5 shows the two CPTs created based on the model shown in Figure 7.4a. The CPT in Figure 7.5a has two configurable nodes while the CPT in Figure 7.5b has eight configurable nodes, all randomly chosen. Notice that we could not create a CPT similar to Figure 7.4a because we are not considering the downgrade operator used in (Buijs, 2014b). Figure 7.6 shows the three CPTs created based on the model shown in Figure 7.4b. The CPT in Figure 7.6a is equivalent to the one shown in Figure 7.4b. The CPT in Figure 7.6b has six configurable nodes while the model in Figure 7.6c has twelve configurable nodes, all randomly chosen. All configurable nodes can be set to enable (or allow), hide or block. We allow each of the three strategies to configure all CPTs to come to the best solution they can achieve. We then compare these results with the results obtained by the ETM on the same dataset.

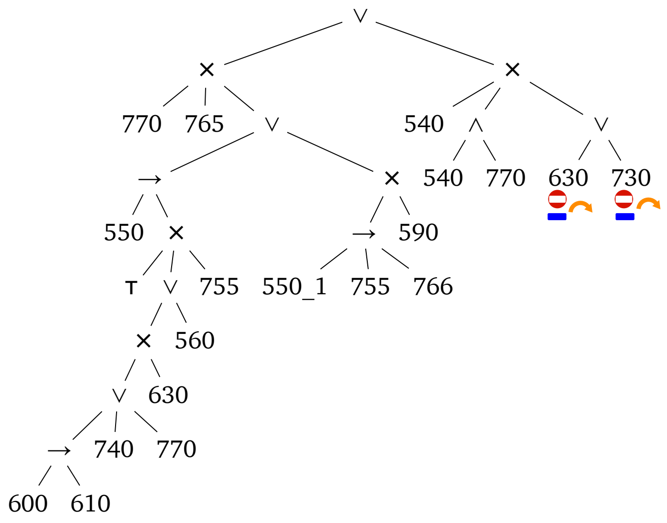


(A) ETM - Approach 3

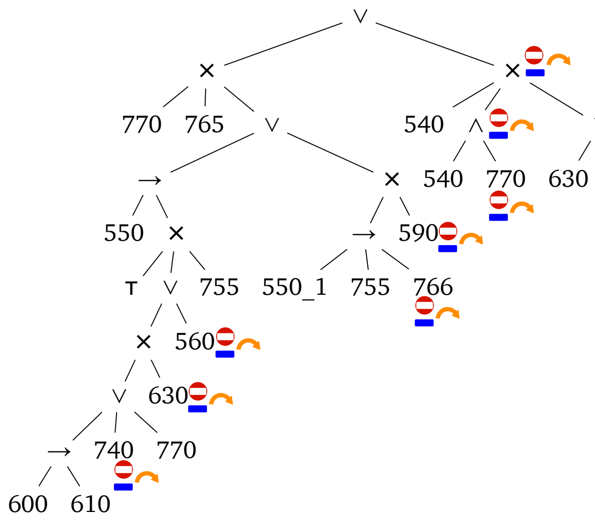


(B) ETM - Approach 4

FIGURE 7.4. Process trees originally discovered by the ETM on the WABO event logs.

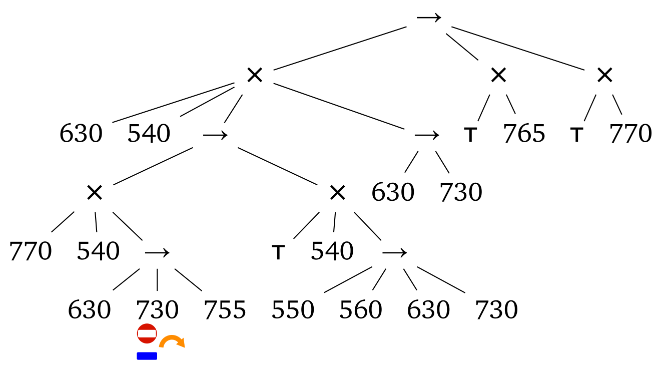


(A) CPT 3-A, contains 2 configurable nodes

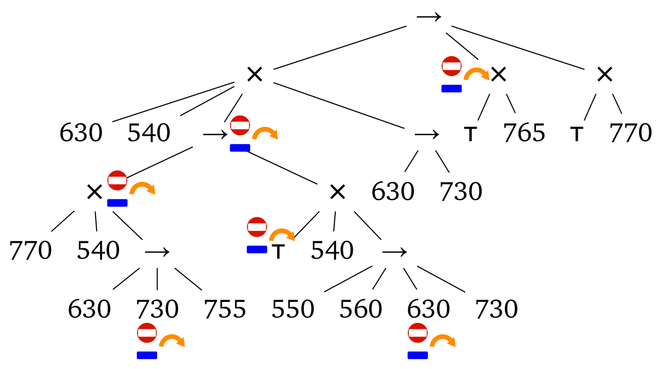


(B) CPT 3-B, contains 8 configurable nodes

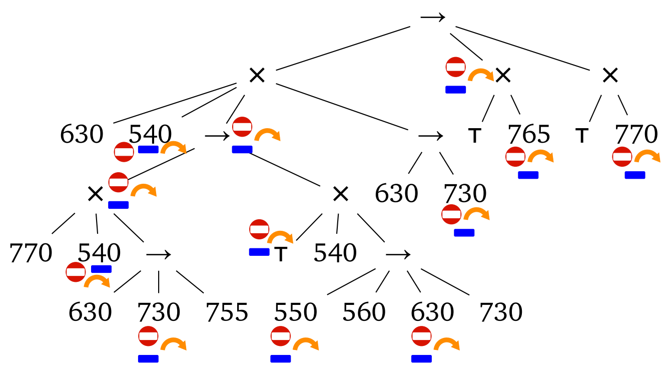
FIGURE 7.5. CPTs based on the model discovered by the ETM Approach 3.



(A) CPT 4-A, contains 1 configurable node



(B) CPT 4-B, contains 6 configurable nodes



(C) CPT 4-C, contains 12 configurable nodes

FIGURE 7.6. CPTs based on the model discovered by the ETM Approach 4.

The results of the experiments with the CPTs created based on the model discovered by the ETM Approach 3 are shown in Table 7.2 and Table 7.3. Table 7.4, Table 7.5 and Table 7.6 show the results of the experiments with the CPTs created based on the model discovered by the ETM Approach 4. In all the tables there are 5 variants of the CPT corresponding to 5 event records of 5 different municipalities. All tables include fitness, precision, and the conformance metric, calculated as 90% fitness and 10% precision.

TABLE 7.2. Results for CPT 3-A and the event logs corresponding to the five process variants

	Variant 1			Variant 2			Variant 3			Variant 4			Variant 5		
	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc
Exhaustive	0.941	0.745	0.921	0.962	0.768	0.943	0.936	0.646	**0.907**	0.985	0.751	0.962	0.950	0.797	0.935
Genetic	0.941	0.745	0.921	0.962	0.768	0.943	0.936	0.646	**0.907**	0.985	0.751	0.962	0.950	0.797	0.935
Greedy	0.878	0.720	0.862	0.908	0.802	0.897	0.855	0.635	0.833	0.973	0.748	0.951	0.907	0.753	0.892
ETM	0.948	0.842	**0.937**	0.948	0.979	**0.951**	0.927	0.729	**0.907**	0.984	0.917	**0.977**	0.957	0.909	**0.952**

f = fitness, p = precision, fpc = conformance

TABLE 7.3. Results for CPT 3-B and the event logs corresponding to the five process variants

	Variant 1			Variant 2			Variant 3			Variant 4			Variant 5		
	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc
Exhaustive	0.958	0.730	0.935	0.962	0.912	0.957	0.945	0.648	**0.915**	0.993	0.849	**0.979**	0.954	0.931	**0.952**
Genetic	0.958	0.730	0.935	0.962	0.912	0.957	0.945	0.648	**0.915**	0.993	0.849	**0.979**	0.954	0.931	**0.952**
Greedy	0.944	0.737	0.923	0.960	0.789	0.943	0.934	0.644	0.905	0.996	0.750	0.971	0.943	0.781	0.927
ETM	0.948	0.842	**0.937**	0.948	0.979	**0.951**	0.927	0.729	0.907	0.984	0.917	0.977	0.957	0.909	**0.952**

f = fitness, p = precision, fpc = conformance

TABLE 7.4. Results for CPT 4-A and the event logs corresponding to the five process variants

	Variant 1			Variant 2			Variant 3			Variant 4			Variant 5		
	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc
Exhaustive	0.913	0.921	**0.914**	0.980	0.962	**0.978**	0.948	0.894	**0.943**	0.986	0.978	**0.985**	0.949	0.975	**0.952**
Genetic	0.913	0.921	**0.914**	0.980	0.962	**0.978**	0.948	0.894	**0.943**	0.986	0.978	**0.985**	0.949	0.975	**0.952**
Greedy	0.913	0.921	**0.914**	0.980	0.962	**0.978**	0.948	0.894	**0.943**	0.985	0.895	0.976	0.949	0.975	**0.952**
ETM	0.913	0.921	**0.914**	0.980	0.962	**0.978**	0.948	0.894	**0.943**	0.986	0.978	**0.985**	0.949	0.975	**0.952**

f = fitness, p = precision, fpc = conformance

TABLE 7.5. Results for CPT 4-B and the event logs corresponding to the five process variants

	Variant 1			Variant 2			Variant 3			Variant 4			Variant 5		
	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc
Exhaustive	0.911	0.951	**0.915**	0.980	0.962	**0.978**	0.948	0.894	**0.943**	0.986	0.978	**0.985**	0.949	0.975	**0.952**
Genetic	0.911	0.951	**0.915**	0.980	0.962	**0.978**	0.948	0.894	**0.943**	0.986	0.978	**0.985**	0.949	0.975	**0.952**
Greedy	0.911	0.951	**0.915**	0.970	0.970	0.970	0.938	0.911	0.935	0.985	0.899	0.976	0.947	0.976	0.950
ETM	0.913	0.921	0.914	0.980	0.962	**0.978**	0.948	0.894	**0.943**	0.986	0.978	**0.985**	0.949	0.975	**0.952**

f = fitness, p = precision, fpc = conformance

TABLE 7.6. Results for CPT 4-C and the event logs corresponding to the five process variants

	Variant 1			Variant 2			Variant 3			Variant 4			Variant 5		
	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc	f	p	fpc
Exhaustive	0.915	0.951	**0.919**	0.980	0.962	**0.978**	0.957	0.894	**0.951**	0.986	0.978	**0.985**	0.949	0.975	**0.952**
Genetic	0.915	0.951	**0.919**	0.980	0.962	**0.978**	0.957	0.894	**0.951**	0.986	0.978	**0.985**	0.949	0.975	**0.952**
Greedy	0.915	0.951	**0.919**	0.952	0.893	0.946	0.948	0.911	0.944	0.975	0.903	0.968	0.945	0.976	0.948
ETM	0.913	0.921	0.914	0.980	0.962	**0.978**	0.948	0.894	0.943	0.986	0.978	**0.985**	0.949	0.975	**0.952**

f = fitness, p = precision, fpc = conformance

It can be observed that in all cases the different strategies obtained desirable range. Since the exhaustive strategy always obtains the best possible result, we can highlight that the genetic strategy always finds the best result, and that the greedy strategy in general obtains very good results, in many cases matching those obtained with the exhaustive strategy.

When comparing the results with those obtained by the ETM, we can point out that in the only case where the CPT is equivalent to the one used by the ETM (Table 7.4 corresponding to CPT 4-A, shown in Figure 7.6a), the results obtained with the ETM and the results obtained with the exhaustive and genetic strategies are the same. On the other hand, the greedy strategy obtained the best results in four of the five variants, except on Variant 4.

When the CPTs have greater flexibility than the CPT used by ETM (CPTs based on ETM Approach 4, CPT 4-B and CPT 4-C, shown in Figure 7.6b and Figure 7.6c, respectively), the search space is larger, so eventually there could be a better configuration. This actually occurs in Variant 1 in Table 7.5, and Variant 1 and Variant 3 in Table 7.6. Moreover, in these highlighted cases, all three strategies are able to find better configurations.

However, in general, the same results were obtained as those obtained by the ETM, as shown for the other variants in Table 7.5 and Table 7.6.

When the CPTs have a different flexibility than the CPT used by ETM (CPTs based on ETM Approach 3, CPT 3-A and CPT 3-B, shown in Figure 7.5a and Figure 7.5b, respectively), the search spaces are not directly comparable. Therefore, in some cases the strategies obtain better results, in other cases the ETM obtains better results, and in others the results are equivalent, as shown in Table 7.2 and Table 7.3.

In this experimental setting, the CPTs are small and the event logs do not contain many variants, therefore when the CPTs do not contain many configurable nodes, the experiments run very fast for all strategies. However, due to the exponential nature of the search space, when the number of configurable nodes grows, the exhaustive strategy may take a long time, such as in CPT 4-C, in which the exhaustive strategy took between 37 minutes (for Variant 3) and almost 3 hours (for Variant 5).

7.3. Algorithms' performance based on an empirical evaluation

To validate the performance of the proposed strategies, a set of controlled experiments was created based on the original reference process model for the universities. The experiments focused on testing the performance of the strategies depending on both the *log complexity* and the *model complexity*.

7.3.1. Performance according to log complexity

The event log complexity depends mainly on the number of trace variants and on how many times each trace variant is repeated in the event log. A trace variant is a particular sequence of activities that can occur multiple times in the event log. This set of experiments evaluates how the algorithms perform under different event log settings: varying the number of trace variants and varying the number of repetitions of each trace variant in the event log. Notice that alignments were computed for each trace variant and reused for all cases of this variant. Therefore, no significant impact is to be expected when varying the number of repetitions of each trace variant.

The configurable process tree contains 9 configurable nodes that have *mixed* dependencies. There are 8 configurable nodes that include $\{H, E\}$ as configurators, and 1 configurable node that has $\{H, B, E\}$ as configurators. The set of feasible configurations allowed by the configurable process model, can be generated based on the Cartesian product, obtaining $2^8 \cdot 3^1 = 768$ feasible configurations. Three different target models (universities *A*, *B* and *C*) derived from the original configurable process model were used for experimentation; the number of activities for the models corresponding to universities *A*, *B* and *C* are 66, 65 and 55, respectively. Based on these target models, different event logs were simulated.

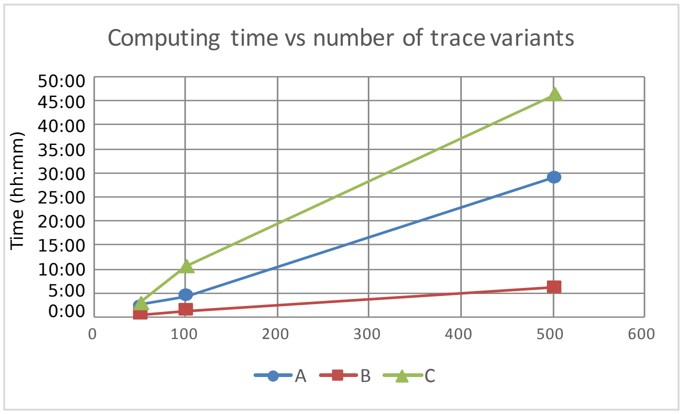
Table 7.7 summarizes the experiments. In the first set of experiments, the number of trace variants is diverse. In this case, a random number of trace repetitions between 10 and 20 is considered for each trace variant. The final number of traces is therefore different for each university. In the second set of experiments, the number of trace variants is fixed to 100, and then the number of trace repetitions is varied. A synthetic event log was created per each experiment, thus 21 event logs were tested with each algorithm. Table 7.8 displays the results obtained with each algorithm when varying the log complexity.

TABLE 7.7. Different log settings for three universities

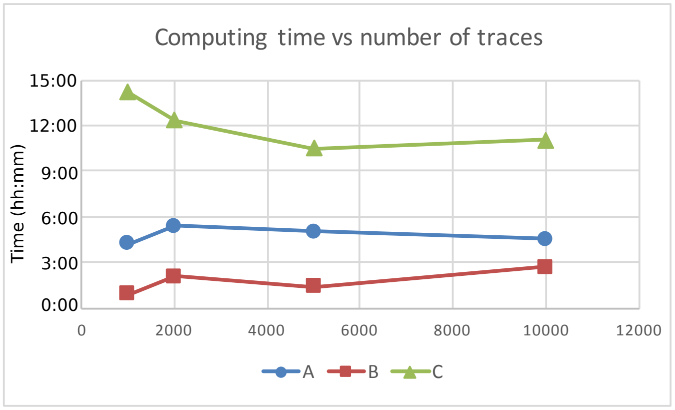
	#trace variants	#trace repetirions	#trace in the event log for a target model A	#trace in the event log for a target model B	#trace in the event log for a target model C
Different trace variants	50	10-20	747	500	500
	100		1466	1543	1000
	500		7448	7456	7450
Different trace repetitions	100	10	1000	1000	1000
		20	2000	2000	2000
		50	5000	5000	5000
		100	10000	10000	10000

7.3.1.1. Exhaustive strategy evaluation

As a brute-force method, this strategy searches over all possible configurations that can be applied to the configurable process model. As seen in Figure 7.7a, computing time increases linearly when increasing the number of trace variants. For event logs with 500 trace variants, the algorithm takes several hours; in the worst case scenario, the university *C*, it takes more than 40 hours to obtain the optimal derived process model. Figure 7.7b shows how the algorithm performs if we set the number of traces to 100 and we vary the number of trace repetitions. In this case, the computing time is not proportional to the log size as when we vary the number of variants; in fact, the computing time is almost constant. This is a feature of the conformance method applied in our implementation to compute the conformance metric. It uses a cache table: if a trace variant has already been evaluated, it is not evaluated again. Therefore, we can observe that the computing time is only proportional to the number of trace variants in the event log, regardless of the number of trace repetitions.



(A) Trace variants complexity

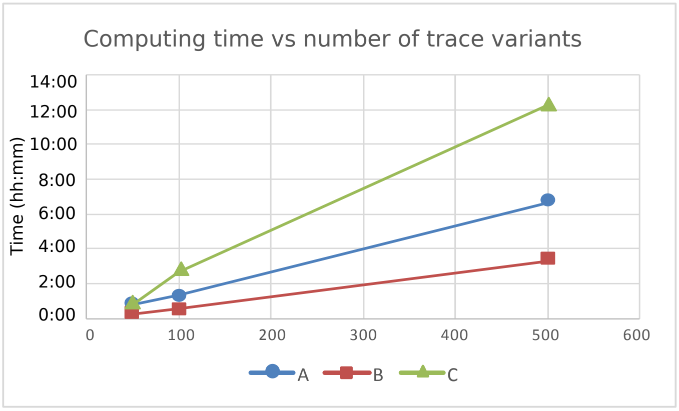


(B) Trace repetitions complexity

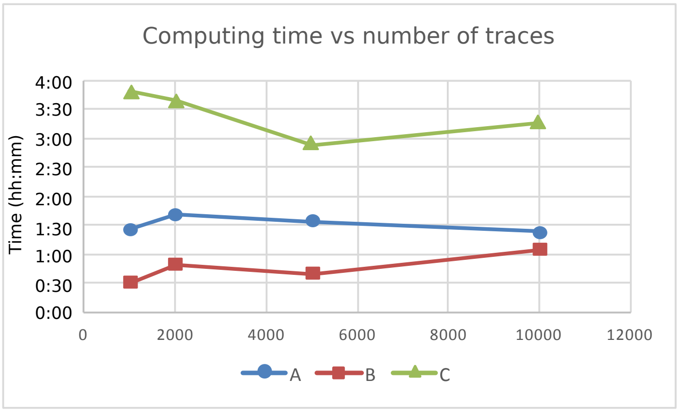
FIGURE 7.7. Performance of the Exhaustive strategy when varying log complexity.

7.3.1.2. Genetic strategy evaluation

As a heuristic method to search for a desirable configuration, the *genetic* method has some parameters. We set the maximum number of generations to 20 and the population size to 10. For all experiments, the *genetic* strategy found an optimal configuration, which allows to obtain an optimal derived process model; in fact, it obtained the same results as the *exhaustive* strategy. Figure 7.8a shows there is a linear dependency between the computing time of the *genetic* strategy and the number of trace variants. Meanwhile, Figure 7.8b depicts the computing time of the *genetic* strategy, which is nearly constant when varying the number of trace repetitions while keeping constant the number of trace variants. As previously mentioned, this is a feature of the conformance method applied to compute the conformance metric.



(A) Trace variants complexity

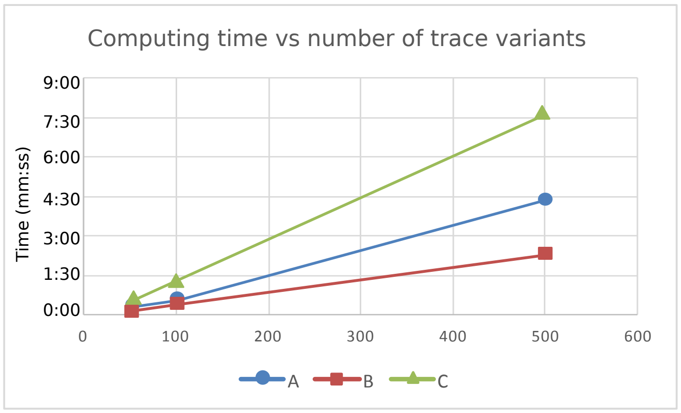


(B) Trace repetitions complexity

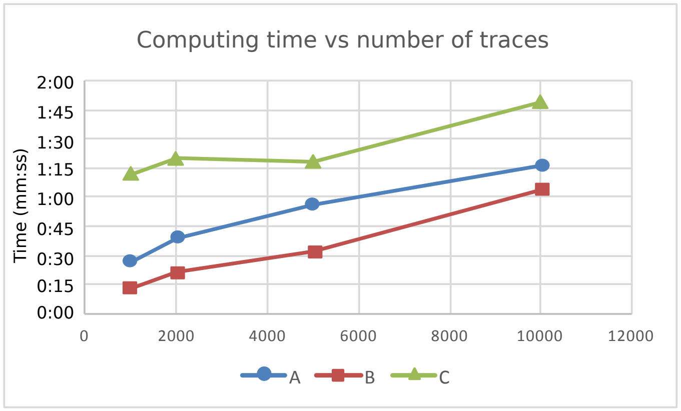
FIGURE 7.8. Performance of the Genetic strategy when varying log complexity.

7.3.1.3. Greedy strategy evaluation

The *greedy* strategy uses a local subtree configuration heuristic to derive the best process subtree for every configurable node. This strategy found a reasonable configuration, and the corresponding derived process model, in all experiments. The solutions are similar to the optimal derived process models obtained by the *exhaustive* and *genetic* strategies. Figure 7.9a shows the computing time varies linearly with the number of trace variants. It also illustrates that the *greedy* algorithm is very fast; it took about 7 minutes in the most complex scenario, corresponding to the university C with 500 trace variants. Figure 7.9b illustrates a slight ascending computing time when increasing the number of trace repetitions while keeping the number of trace variants constant.



(A) Trace variants complexity



(B) Trace repetitions complexity

FIGURE 7.9. Performance of the Greedy strategy when varying the log complexity.

In summary, the *greedy* strategy finds a reasonable derived process model, which is very close to the optimal process model in a short computing time, whereas the *exhaustive* and *genetic* strategies find an optimal derived process model, but require more computing time. In all cases, the performance of the algorithms vary linearly with the number of trace variants, and it does not depend on the number of trace repetitions. In addition, Table 7.8 shows that the genetic strategy obtains the same conformance as the exhaustive strategy in all experiments for the three universities, except in one case (100t 10rep), in which the results are quite similar. By contrast, the conformance of the greedy for University C is below the conformance obtained by the other two strategies. The strategies applied to the event log of the University C do not reach 0.8 of conformance, meaning that providing a model for this location could be reconsidered.

TABLE 7.8. Results obtained for all algorithms considering different log complexity.

		A		B		C	
Strategy	Log complexity	time (hh:mm:ss)	conformance	time (hh:mm:ss)	conformance	time (hh:mm:ss)	conformance
Exhaustive	50t 10-20rep	02:28:23	0,922	00:27:36	0,939	02:41:04	0,672
	100t 10-20rep	04:08:03	0,923	01:10:33	0,935	10:04:35	0,784
	500t 10-20rep	27:40:27	0,924	05:55:05	0,941	44:14:38	0,787
	100t 10rep	04:04:22	0,923	00:50:08	0,939	13:36:03	0,782
	100t 20rep	05:10:53	0,921	02:00:30	0,939	11:49:07	0,783
	100t 50rep	04:50:08	0,924	01:20:34	0,939	10:04:29	0,788
	100t 100rep	04:21:51	0,924	02:32:39	0,935	10:36:01	0,784
Genetic	50t 10-20rep	00:56:10	0,922	00:16:08	0,939	00:56:43	0,672
	100t 10-20rep	01:34:54	0,923	00:41:19	0,935	03:13:04	0,784
	500t 10-20rep	07:57:34	0,924	03:56:26	0,941	14:40:04	0,787
	100t 10rep	01:21:56	0,923	00:28:26	0,939	03:39:13	0,781
	100t 20rep	01:37:05	0,921	00:47:12	0,939	03:30:46	0,783
	100t 50rep	01:30:28	0,924	00:37:23	0,939	02:46:14	0,788
	100t 100rep	01:20:15	0,924	01:01:52	0,935	03:07:46	0,784
Greedy	50t 10-20rep	00:00:18	0,920	00:00:08	0,938	00:00:29	0,671
	100t 10-20rep	00:00:32	0,920	00:00:23	0,933	00:01:16	0,784
	500t 10-20rep	00:04:10	0,922	00:02:11	0,940	00:07:19	0,786
	100t 10rep	00:00:30	0,920	00:00:14	0,938	00:01:22	0,780
	100t 20rep	00:00:45	0,919	00:00:25	0,937	00:01:32	0,782
	100t 50rep	00:01:05	0,921	00:00:37	0,938	00:01:30	0,787
	100t 100rep	00:01:28	0,922	00:01:13	0,934	00:02:06	0,784

7.3.2. Performance according to model complexity

Process model derivation does not only depend on the log complexity, but also on the model complexity. This set of experiment evaluates how the strategies perform under different configurable process model topologies and different configurable node dependencies. Based on the original configurable process model, three configurable process models were built with different topological characteristics: models that only include $\times$ and $\wedge$ operator nodes, models that only include $\times$ and $\circlearrowleft$ operator nodes, and models that consider all operator nodes ($\times$, $\wedge$ and $\circlearrowleft$). In addition, for each one of these three configurable process models, three different configurable node dependencies were created: *independent*, *dependent* and *mixed* (as described in section 5.3). In total, there are 9 different configurable process models Q_i^{α}, where $i = 1, \ldots, 9$; they are summarized in Table 7.9.

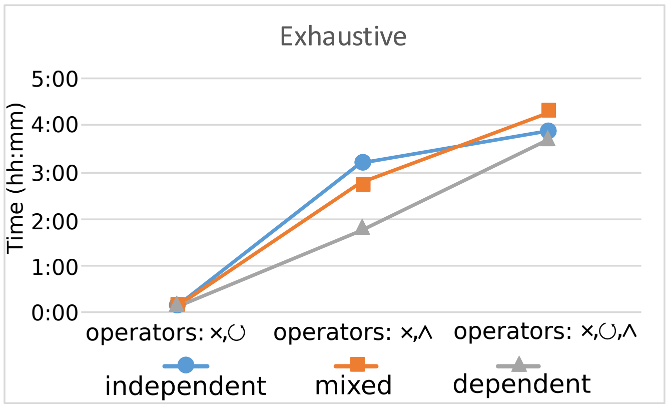
Three different target models derived from each of these 9 configurable process models were used for experimentation. Based on these 27 target models, 27 different event logs were simulated. In Table 7.9, they are represented as L_{i_1}, L_{i_2} and L_{i_3}, for $i = 1, \ldots, 9$.

We applied the *exhaustive*, *genetic*, and *greedy* strategies to all 9 configurable process models with their corresponding 3 event logs. The results are shown in Table 7.9, including the computing time and the conformance metric. The computing time required for all strategies is shown in Figure 7.10, where the average computing time required for the 3 event logs created for each configurable process model is displayed.

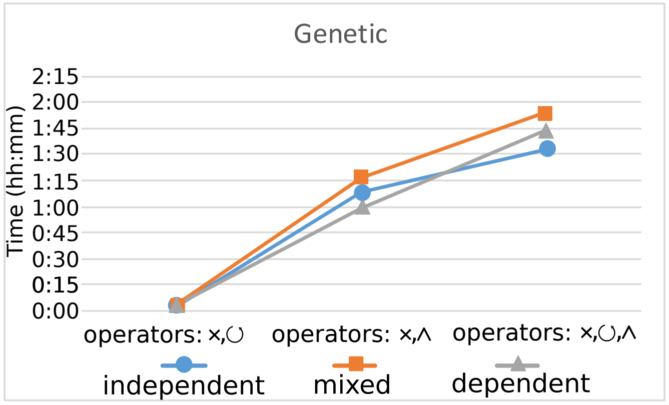
Figure 7.10a depicts how much time requires the *exhaustive* strategy to evaluate all possible configurations in order to derive the best process tree in each case. The results obtained suggest that if the configurable process model has parallelism, the computing time required to get a derived process model increases, such as in the case of models with topologies that contain $\times$ and $\wedge$ operators or $\times$, $\wedge$ and $\circlearrowleft$ operators. This is due to the conformance method requires more time to handle parallelism, because the traces to be processed are usually longer (Munoz-Gama, 2016). Whereas models with a topology that only contains $\times$ and $\circlearrowleft$ operators require a lower computing time to derive a process model. It can be noticed that this is independent of the configurable node dependencies.

A similar behavior can be observed in Figure 7.10b for the *genetic* strategy and in Figure 7.10c for the *greedy* strategy. In all cases, more computing time is required when the configurable process models have parallelism, regardless of the configurable nodes dependencies.

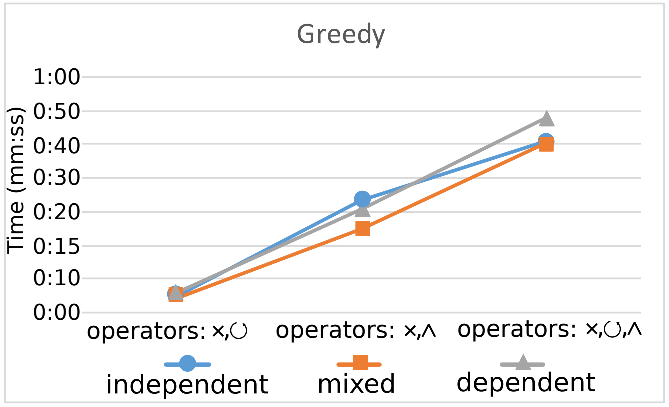
Figure 7.10c also depicts that the *greedy* strategy to derive a process model, requires considerable less computing time than the other two strategies.



(A) Exhaustive strategy



(B) Genetic strategy



(C) Greedy strategy

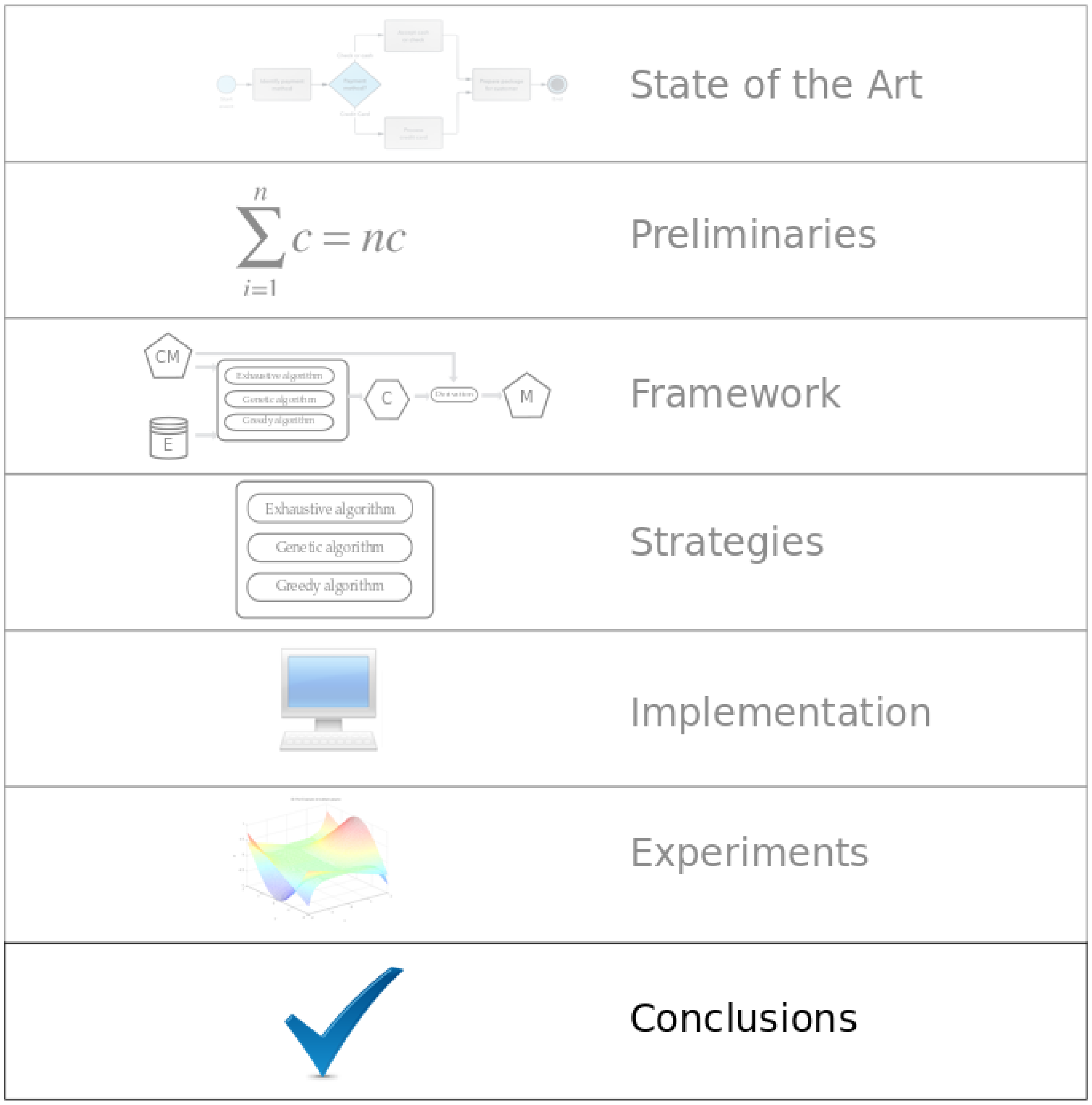
FIGURE 7.10. Performance of the different strategies when varying the model complexity.

TABLE 7.9. Results obtained for all strategies considering different model complexity.

				Strategies					
				Exhaustive		Genetic		Greedy	
	Configurable model topology	Configurable Node dependencies	Event log	time (hh:mm:ss)	conformance	time (hh:mm:ss)	conformance	time (hh:mm:ss)	conformance
			L_{1_1}	00:08:15	0,98	00:01:46	0,98	00:00:04	0,98
Q_1^α		Independent	L_{1_2}	00:06:43	0,99	00:01:03	0,99	00:00:02	0,98
			L_{1_3}	00:12:47	0,84	00:02:42	0,84	00:00:05	0,84
			L_{2_1}	00:07:05	0,98	00:01:35	0,98	00:00:03	0,97
Q_2^α	operators: $\times$,$\circlearrowleft$	Mixed	L_{2_2}	00:05:25	0,98	00:01:19	0,98	00:00:02	0,97
			L_{2_3}	00:09:17	0,84	00:02:49	0,84	00:00:05	0,83
			L_{3_1}	00:07:51	0,98	00:02:15	0,98	00:00:05	0,98
Q_3^α		Dependent	L_{3_2}	00:06:20	0,98	00:01:56	0,98	00:00:04	0,98
			L_{3_3}	00:08:18	0,89	00:03:37	0,89	00:00:06	0,87
			L_{4_1}	03:47:51	0,93	01:01:16	0,93	00:00:25	0,93
Q_4^α		Independent	L_{4_2}	02:54:15	0,95	00:45:41	0,95	00:00:14	0,94
			L_{4_3}	04:49:52	0,87	01:29:55	0,87	00:00:47	0,87
			L_{5_1}	02:43:11	0,95	01:06:10	0,95	00:00:19	0,95
Q_5^α	operators: $\times$, $\wedge$	Mixed	L_{5_2}	01:57:22	0,95	00:48:31	0,95	00:00:09	0,95
			L_{5_3}	05:18:59	0,89	01:44:15	0,89	00:00:36	0,89
			L_{6_1}	01:59:09	0,94	00:49:10	0,94	00:00:24	0,94
Q_6^α		Dependent	L_{6_2}	01:32:55	0,96	00:37:14	0,96	00:00:12	0,96
			L_{6_3}	02:49:35	0,87	01:24:52	0,87	00:00:43	0,89
			L_{7_1}	04:32:13	0,93	01:02:33	0,93	00:00:31	0,92
Q_7^α		Independent	L_{7_2}	01:32:53	0,93	00:48:54	0,93	00:00:26	0,93
			L_{7_3}	07:52:15	0,82	02:36:41	0,82	00:01:16	0,81
			L_{8_1}	04:08:03	0,92	01:34:54	0,92	00:00:32	0,92
Q_8^α	operators: $\times$, $\wedge$, $\circlearrowleft$	Mixed	L_{8_2}	01:10:33	0,94	00:41:19	0,94	00:00:23	0,93
			L_{8_3}	10:04:35	0,78	03:13:04	0,78	00:01:16	0,78
			L_{9_1}	03:55:53	0,92	01:35:44	0,92	00:00:44	0,88
Q_9^α		Dependent	L_{9_1}	01:05:22	0,92	00:35:24	0,92	00:00:27	0,91
			L_{9_3}	08:12:41	0,83	02:50:31	0,83	00:01:20	0,82

8. CONCLUSIONS

In this chapter we summarize our findings and present the improvements that have to be done as future work. We also present the limitations of this research.



8.1. General Conclusions

In this thesis, we propose a framework to derive a process tree from a configurable process tree based on the historical behavior of a process stored in a given event log. Through three different strategies, *exhaustive*, *genetic* and *greedy*, we allow to derive a process model that better conforms to the event log. The *exhaustive* strategy searches among all possible derived process models given by the Cartesian product of all possible configurations. This strategy finds an optimal configuration, which is used to derive the best possible process model, but not in a suitable time. The *genetic* strategy also finds a quasi-optimal configuration (sometimes even the optimal one) to derive a process model, but in less time than the *exhaustive* approach. Although in a general case optimality is not guaranteed, in our experiments it was always able to find an optimal configuration. Moreover, in our experiments the *genetic* strategy did not have to iterate for many generations; it converges in less than 20 generations. Meanwhile, the *greedy* strategy finds a good approximate configuration to derive a process model in a very short time compared to the other two algorithms. We have validated the applicability of our framework using both realistic and real life processes. The proposed strategies allow to find a very good derived process model in a short time, using the *greedy* strategy, or an optimal derived process model using the *exhaustive* or the *genetic* strategies.

Table 8.1 summarizes van der Aalst's findings and also shows our contribution to the state-of-the-art in configurable process models.

TABLE 8.1. Weaknesses identified in the 289 papers analyzed by van der Aalst (2012) and how they are overcome in this thesis.

Weakness	289 BPM papers	This thesis
Many papers introduce a new modeling language	Only used for the paper or the need is not justified	We use an existing modeling language: process tree
Many papers cannot be linked to one of the 20 use cases found on Figure 2.1	Authors seem to focus on originality rather than relevance and show little concern for real-life use cases	We present a new composite use case, called ConCMEV
Many papers describe implementation effort	In many cases, implementation is not available	We implement our strategies on ProM, which is a publicly available process mining tool
Many papers include case studies	In many cases the case studies are artificial	We address both a realistic use case and a real life use case

8.2. Limitations

Our work is restricted to the new use case presented in chapter 1 that seeks to obtain a configuration to derive a process model from a configurable process model which maximizes the conformance function for a given event log. In this context, our research presents the following limitations. First, a configurable process model that contains independent configurable nodes must be available, including a set of feasible configurations for each configurable node. Second, the event log must be representative of the executed process, i.e., it must contain traces representing all possible behaviors of the process; otherwise, the unobserved behavior could be left out of the derived process model. Notice that this is a common limitation of all process discovery techniques used in process mining. Third, each strategy has its own limitations, since the three proposed strategies seek to balance two opposing goals: to obtain optimal solutions versus to obtain solutions quickly. As observed in the analysis of chapter 7, the exhaustive strategy allows to obtain an optimal solution, but taking a considerable time. In contrast, the greedy strategy allows to obtain a solution quickly, although not necessarily an optimal one. An intermediate approach is the genetic strategy, which allows to obtain a very good—even potentially optimal—solution in a shorter time than the exhaustive strategy. Finally, the three strategies seek to find a single process model that maximizes the conformance function, which in turn weighs the four quality criteria. In contrast, there are discovery techniques that use Pareto Front (Buijs, 2014b) to obtain a set of process models that represent the best possible solutions for different weights of the four quality criteria. However, this work does not consider it.

The proposed approach does not address privacy issues that usually emerge in cross-organizational business process settings (Liu et al., 2016). We assume both the configurable process model and the event log belong to the same organization, which might be a corporate group that wants to have a common reference model (a configurable process model) for a given process among all the companies it owns, and also might be interested in obtaining a derived process model for a (new) company by using the approach proposed in this thesis.

8.3. Future Work

The main future works that can be performed to extend this research are following underlined. First of all, user-defined rules can be used to guide the configuration process (Schunselaar et al., 2014). These rules can constrain the search space, reducing the computing time required to derive a process model. Second, large configurable process models could be configured in successive stages through a decomposition approach (de Leoni, Munoz-Gama, Carmona, & van der Aalst, 2014). When a configurable process model is large, it can be decomposed in sub-process models that can be configured independently, in order to reduce complexity and improve performance. Third, on the greedy strategy, we used a bottom-up approach. A top-down approach could also be explored. Fourth, including operator downgrading as another way to configure a node. By downgrading an operator, the behavior of the operator is restricted to a subset of the initially possible behavior (Buijs, 2014b). Finally, when the conformance obtained by the derived process model is not good enough, you might think the event log contains knowledge about the process that is not considered in the configurable process model (e.g., activities that are observed in the event log, but do not exist in the configurable process model). In that case, you could use the event log to first enrich the configurable process model, before tackling the derivation task.

References

Adriansyah, A. (2014). *Aligning Observed and Modeled Behavior* (Unpublished doctoral dissertation). Eindhoven University of Technology.

Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B. F., & van der Aalst, W. M. (2012). Alignment based precision checking. In *International conference on business process management* (pp. 137–149).

Arriagada-Benítez, M., Sepúlveda, M., Munoz-Gama, J., & Buijs, J. C. (2017). Strategies to automatically derive a process model from a configurable process model based on event data. *Applied Sciences*, *7*(10), 1023.

Bäck, T., de Graaf, J. M., Kok, J. N., & Kosters, W. A. (2001). Theory of genetic algorithms. In *Current trends in theoretical computer science* (pp. 546–578).

Becker, J., Delfmann, P., Dreiling, A., Knackstedt, R., & Kuropka, D. (2004). Configurative process modeling–outlining an approach to increased business process model usability. In *Proceedings of the 15th irma international conference* (pp. 1–12).

Becker, J., Delfmann, P., & Knackstedt, R. (2007). Adaptive reference modeling: integrating configurative and generic adaptation techniques for information models. *Reference Modeling*, 27–58.

Buijs, J. C. A. M. (2014a). *Environmental permit application process (wabo), coselog project.* https://data.4tu.nl/repository/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270. Eindhoven University of Technology. doi: 10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

Buijs, J. C. A. M. (2014b). *Flexible Evolutionary Algorithms for Mining Structured Process Models* (Unpublished doctoral dissertation). Eindhoven University of Technology.

Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2012a). A genetic algorithm for discovering process trees. In *Proceedings of the IEEE congress on evolutionary computation, CEC 2012, brisbane, australia, june 10-15, 2012* (pp. 1–8).

Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2012b). On the role of fitness, precision, generalization and simplicity in process discovery. In R. Meersman et al. (Eds.), *OTM 2012* (Vol. 7565, pp. 305–322). Springer.

Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2013). Mining configurable process models from collections of event logs. In F. Daniel, J. Wang, & B. Weber (Eds.), *Bpm 2013* (Vol. 8094, pp. 33–48). Springer.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms (3. ed.)*. MIT Press.

Deb, K., & Deb, D. (2014). Analysing mutation schemes for real-parameter genetic algorithms. *IJAISC*, *4*(1), 1–28.

de Leoni, M., Munoz-Gama, J., Carmona, J., & van der Aalst, W. M. P. (2014). Decomposing alignment-based conformance checking of data-aware process models. In *OTM conferences* (Vol. 8841, pp. 3–20). Springer.

Derguech, W., Bhiri, S., & Curry, E. (2017). Designing business capability-aware configurable process models. *Information Systems*, *72*, 77–94.

Dijkman, R. M., Rosa, M. L., & Reijers, H. A. (2012). Managing large collections of business process models - current techniques and challenges. *Computers in Industry*, *63*(2), 91–97.

Dumas, M., Van der Aalst, W. M., & Ter Hofstede, A. H. (2005). *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons.

Ghedira, C., & Mezni, H. (2006). Through personalized web service composition specification: from bpel to c-bpel. *Electronic Notes in Theoretical Computer Science*, *146*(1), 117–132.

Gottschalk, F. (2009). *Configurable process models* (Unpublished doctoral dissertation). Eindhoven University of Technology.

Gottschalk, F., van der Aalst, W. M., & Jansen-Vullers, M. H. (2007). Configurable process models –a foundational approach. In *Reference modeling* (pp. 59–77). Springer.

Gottschalk, F., van der Aalst, W. M. P., & Jansen-Vullers, M. H. (2008a). Merging event-driven process chains. In *OTM conferences (1)* (Vol. 5331, pp. 418–426). Springer.

Gottschalk, F., van der Aalst, W. M. P., & Jansen-Vullers, M. H. (2008b). Mining reference process models and their configurations. In *OTM workshops* (Vol. 5333, pp. 263–272). Springer.

Gottschalk, F., van der Aalst, W. M. P., Jansen-Vullers, M. H., & Rosa, M. L. (2008). Configurable workflow models. *Int. J. Cooperative Inf. Syst.*, *17*(2), 177–221.

Gottschalk, F., Wagemakers, T. A. C., Jansen-Vullers, M. H., van der Aalst, W. M. P., & Rosa, M. L. (2009). Configurable process models: Experiences from a municipality case study. In *Advanced information systems engineering, 21st international conference, caise 2009, proceedings* (pp. 486–500).

Günther, C. W., & van der Aalst, W. M. P. (2007). Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In *BPM* (Vol. 4714, pp. 328–343). Springer.

Jansen-Vullers, M. H., van der Aalst, W. M. P., & Rosemann, M. (2006). Mining configurable enterprise information systems. *Data Knowl. Eng.*, *56*(3), 195–244.

Lee, C., Choy, K. L., Ho, G. T., & Lam, C. H. (2016). A slippery genetic algorithm-based process mining system for achieving better quality assurance in the garment industry. *Expert Systems with Applications*, *46*, 236–248.

Leemans, S. J. J., Fahland, D., & van der Aalst, W. M. P. (2013a). Discovering block-structured process models from event logs - A constructive approach. In J. M. Colom & J. Desel (Eds.), *PETRI NETS 2013* (Vol. 7927, pp. 311–329). Springer.

Leemans, S. J. J., Fahland, D., & van der Aalst, W. M. P. (2013b). Discovering block-structured process models from event logs containing infrequent behaviour. In N. Lohmann, M. Song, & P. Wohed (Eds.), *Bpm 2013* (Vol. 171, pp. 66–78). Springer.

Leemans, S. J. J., Fahland, D., & van der Aalst, W. M. P. (2014). Discovering block-structured process models from incomplete event logs. In *Petri nets* (Vol. 8489, pp. 91–110). Springer.

Liu, C., Duan, H., Qingtian, Z., Zhou, M., Lu, F., & Cheng, J. (2016). Towards comprehensive support for privacy preservation cross-organization business process mining. *IEEE Transactions on Services Computing*.

Meffert, K., Rotstan, N., Knowles, C., & Sangiorgi, U. (2008). Jgap-java genetic algorithms and genetic programming package. *URL: http://jgap. sf. net*.

Mendling, J., & Simon, C. (2006). Business process design by view integration. In J. Eder & S. Dustdar (Eds.), *Bpm 2006* (Vol. 4103, pp. 55–64). Springer.

Michalewicz, Z. (1996). *Genetic algorithms and data structures - evolution programs (3. ed.)*. Springer.

Mitchell, M. (1998). *An introduction to genetic algorithms*. Cambridge, MA, USA: MIT Press.

Munoz-Gama, J. (2016). *Conformance checking and diagnosis in process mining - comparing observed and modeled processes* (Vol. 270). Springer.

Munoz-Gama, J., Carmona, J., & van der Aalst, W. M. P. (2013). Conformance checking in the large: Partitioning and topology. In *BPM* (Vol. 8094, pp. 130–145). Springer.

Munoz-Gama, J., Carmona, J., & van der Aalst, W. M. P. (2014). Single-entry single-exit decomposed conformance checking. *Inf. Syst.*, *46*, 102–122.

Munoz-Gama, J. (2014). *Conformance Checking and Diagnosis in Process Mining* (Unpublished doctoral dissertation). Universitat Politècnica de Catalunya.

Rosa, M. L., Dumas, M., ter Hofstede, A. H. M., & Mendling, J. (2011). Configurable multi-perspective business process models. *Inf. Syst.*, *36*(2), 313–340.

Rosa, M. L., Dumas, M., ter Hofstede, A. H. M., Mendling, J., & Gottschalk, F. (2008). Beyond control-flow: Extending business process configuration to roles and objects. In *ER* (Vol. 5231, pp. 199–215). Springer.

Rosa, M. L., Dumas, M., Uba, R., & Dijkman, R. M. (2010). Merging business process models. In R. Meersman, T. S. Dillon, & P. Herrero (Eds.), *On the move to meaningful internet systems: OTM 2010* (Vol. 6426, pp. 96–113). Springer.

Rosa, M. L., Dumas, M., Uba, R., & Dijkman, R. M. (2013). Business process model merging: An approach to business process consolidation. *ACM Trans. Softw. Eng. Methodol.*, *22*(2), 11.

Rosa, M. L., & Gottschalk, F. (2009). Synergia - comprehensive tool support for configurable process models. In A. K. A. de Medeiros & B. Weber (Eds.), *Bpmdemos 2009* (Vol. 489). CEUR-WS.org.

Rosa, M. L., van der Aalst, W. M. P., Dumas, M., & ter Hofstede, A. H. M. (2009). Questionnaire-based variability modeling for system configuration. *Software and System Modeling*, *8*(2), 251–274.

Rosemann, M., & van der Aalst, W. M. P. (2007). A configurable reference modelling language. *Inf. Syst.*, *32*(1), 1–23.

Rozinat, A., & van der Aalst, W. M. P. (2005). Conformance testing: Measuring the fit and appropriateness of event logs and process models. In *Business process management workshops* (Vol. 3812, pp. 163–176).

Rozinat, A., & van der Aalst, W. M. P. (2008). Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, *33*(1), 64–95.

Safe, M. D., Carballido, J. A., Ponzoni, I., & Brignole, N. B. (2004). On stopping criteria for genetic algorithms. In A. L. C. Bazzan & S. Labidi (Eds.), *Advances in artificial intelligence - SBIA 2004* (Vol. 3171, pp. 405–413). Springer.

Schunselaar, D. M. M., Leopold, H., Verbeek, H. M. W., van der Aalst, W. M. P., & Reijers, H. A. (2014). Configuring configurable process models made easier: An automated approach. In F. Fournier & J. Mendling (Eds.), *Bpm 2014* (Vol. 202, pp. 105–117). Springer.

Schunselaar, D. M. M., Verbeek, E., van der Aalst, W. M. P., & Reijers, H. A. (2012). Creating sound and reversible configurable process models using cosenets. In W. Abramowicz, D. Kriksciuniene, & V. Sakalauskas (Eds.), *Bis 2012* (Vol. 117, pp. 24–35). Springer.

Seidel, S., Rosemann, M., Hofstede, A. t., & Bradford, L. (2006). Developing a business process reference model for the screen business−a design science research case study. *ACIS 2006 Proceedings*, 39.

Sharma, D. K., & Rao, V. (2014). Configurable business process modeling notation. In *Advance computing conference (iacc), 2014 ieee.* (pp. 1424–1429).

Tiwari, A., Turner, C., & Majeed, B. (2008). A review of business process mining: state−of−the−art and future trends. *Business Process Management Journal*, *14*(1), 5-22. doi: 10.1108/14637150810849373 vanden Broucke, S. K. L. M., De Weerdt, J., Baesens, B., & Vanthienen, J. (2012). Improved artificial negative event generation to enhance process event logs. In J. Ralyté, X. Franch, S. Brinkkemper, & S. Wrycza (Eds.), *Advanced information systems engineering: 24th international conference, caise 2012, gdansk, poland, june 25-29, 2012. proceedings* (pp. 254–269). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from `https://doi.org/10.1007/978-3-642-31095-9_17` doi: 10.1007/978-3-642-31095-9_17

van der Aalst, W. M. P. (2011a). *Process mining - discovery, conformance and enhancement of business processes*. Springer.

van der Aalst, W. M. P. (2011b). Using process mining to bridge the gap between BI and BPM. *IEEE Computer*, *44*(12), 77–80.

van der Aalst, W. M. P. (2012). A Decade of Business Process Management Conferences: Personal Reflections on a Developing Discipline. In A. P. Barros, A. Gal, & E. Kindler (Eds.), *Bpm 2012* (Vol. 7481, pp. 1–16). Springer.

van der Aalst, W. M. P. (2013a). Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, *2013*.

van der Aalst, W. M. P. (2013b). Decomposing petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, *31*(4), 471–507. doi: 10.1007/s10619-013-7127-5

van der Aalst, W. M. P. (2016). *Process mining - data science in action, second edition*. Springer.

van der Aalst, W. M. P., Buijs, J. C. A. M., & van Dongen, B. F. (2011). Towards improving the representational bias of process mining. In K. Aberer, E. Damiani, & T. S. Dillon (Eds.), *SIMPDA 2011* (Vol. 116, pp. 39–54). Springer.

van der Aalst, W. M. P., Dreiling, A., Gottschalk, F., Rosemann, M., & Jansen-Vullers, M. H. (2005). Configurable process models as a basis for reference modeling. In C. Bussler & A. Haller (Eds.), *BPM 2005* (Vol. 3812, pp. 512–518).

van der Aalst, W. M. P., Dumas, M., Ouyang, C., Rozinat, A., & Verbeek, E. (2008). Conformance checking of service behavior. *ACM Trans. Internet Techn.*, *8*(3), 13:1–13:30.

van der Aalst, W. M. P., Lohmann, N., & Rosa, M. L. (2012). Ensuring correctness during process configuration via partner synthesis. *Inf. Syst.*, *37*(6), 574–592.

van der Aalst, W. M. P., Lohmann, N., Rosa, M. L., & Xu, J. (2010). Correctness ensuring process configuration: An approach based on partner synthesis. In R. Hull, J. Mendling, & S. Tai (Eds.), *Bpm 2010* (Vol. 6336, pp. 95–111). Springer.

van der Aalst, W. M. P., van Dongen, B. F., Günther, C. W., Rozinat, A., Verbeek, E., & Weijters, T. (2009). Prom: The process mining toolkit. In *BPM (demos)* (Vol. 489). CEUR-WS.org.

van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., & Weijters, A. J. M. M. (2003). Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.*, *47*(2), 237–267.

van der Aalst, W. M. P., van Hee, K. M., van der Werf, J. M. E. M., & Verdonk, M. (2010). Auditing 2.0: Using process mining to support tomorrow's auditor. *IEEE Computer*, *43*(3), 90–93.

van der Aalst, W. M. P., & Weijters, A. J. M. M. (2004). Process mining: a research agenda. *Computers in Industry*, *53*(3), 231–244.

van der Aalst, W. M. P., Weijters, T., & Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, *16*(9), 1128–1142.

Van Dongen, B. (2015). *Bpi challenge 2015.* http://dx.doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1. Eindhoven University of Technology. doi: 10.4121/uuid:31a308ef-c844-48da-948c -305d167a0ec1

van Oirschot, Y. (2014). *Using trace clustering for configurable process discovery explained by event log data* (Unpublished master's thesis). Eindhoven University of Technology.

Vázquez-Barreiros, B., Mucientes, M., & Lama, M. (2015). Prodigen: Mining complete, precise and minimal structure process models with a genetic algorithm. *Information Sciences*, *294*, 315–333.

Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2010). Xes, xesame, and prom 6. In *Caise forum* (Vol. 72, pp. 60–75). Springer.

Weijters, A., Aalst, W. M. P., & A K Medeiros, A. (2006). *Process mining with the heuristics miner-algorithm* (Vol. 166).

Weijters, A. J. M. M., & Ribeiro, J. T. S. (2011). Flexible heuristics miner (FHM). In *Computational intelligence and data mining (cidm), 2011 ieee symposium on* (pp. 310–317). IEEE.

Wolffensperger, R. (2014). *Static and dynamic visualization of quality and performance dimensions on process trees* (Unpublished master's thesis). Eindhoven University of Technology.

APPENDIX A. ACADEMIC PLANNING

Figure A.1 represents the overview of the Academic Planning full model containing the 70 activities.

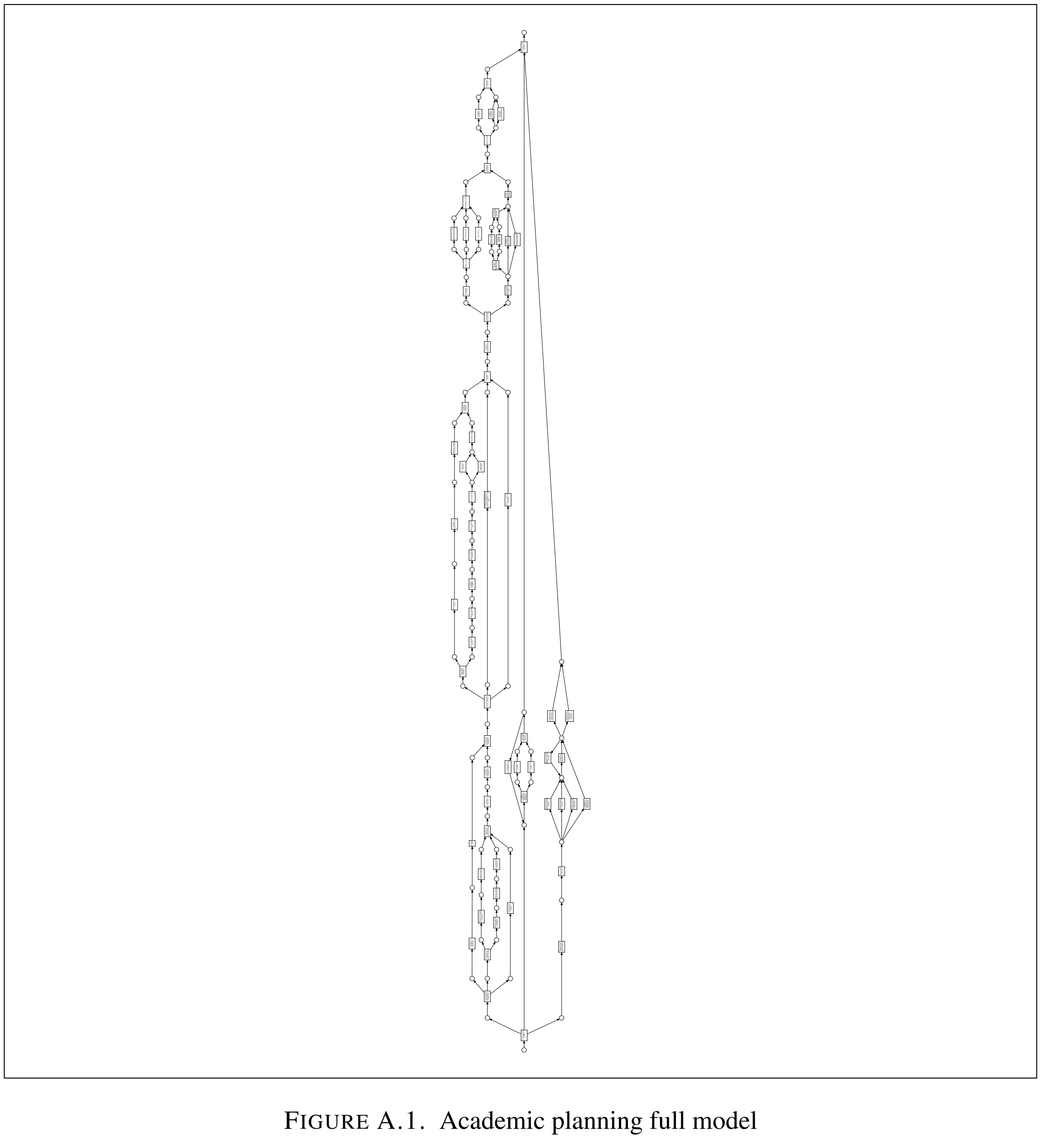


FIGURE A.1. Academic planning full model

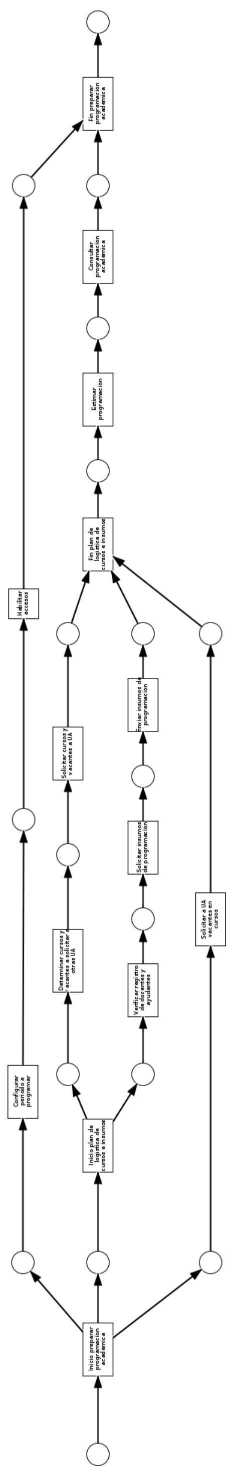

Inicio preparar programación académica
Configurar periodo a programar
Inicio plan de logística de cursos e insumos
Determinar cursos y vacantes a solicitar a otras UA
Verificar registro de docentes y ayudantes
Solicitar a UA vacantes en cursos
Solicitar insumos de programación
Solicitar cursos y vacantes a UA
Enviar insumos de programación
Habilitar accesos
Fin plan de logística de cursos e insumos
Estimar programación
Consultar programación académica
Fin preparar programación académica

FIGURE A.2. Prepare Academic Program sub-model

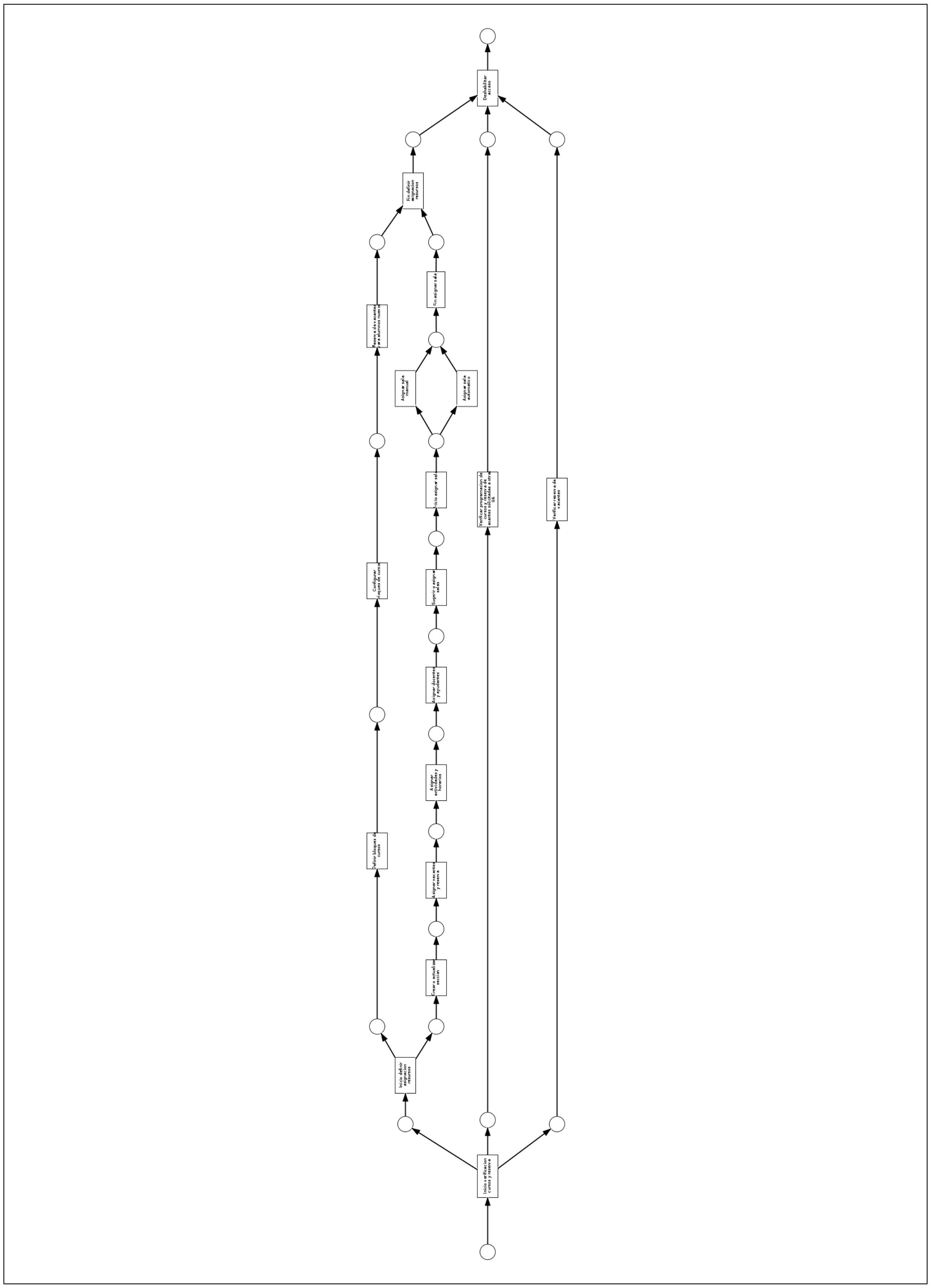

Inicio verificacion cursos y reserva
Inicio definir asignacion resursos
Crear o actualizar seccion
Asignar vacantes y reserva
Asignar actividades y horarios
Asignar docentes y ayudantes
Sugerir o asignar salas
Inicio asignar sala
Asignar sala manual
Asignar sala automatico
Fin asignar sala
Definir bloques de cursos
Configurar bloques de cursos
Reserva de vacantes para alumnos nuevos
Fin definir asignacion resursos
Verificar programacion de cursos y reserva de vacantes solicitadas a otras UA
Verificar reserva de vacantes
Deshabilitar acceso

FIGURE A.3. Update Academic Program sub-model

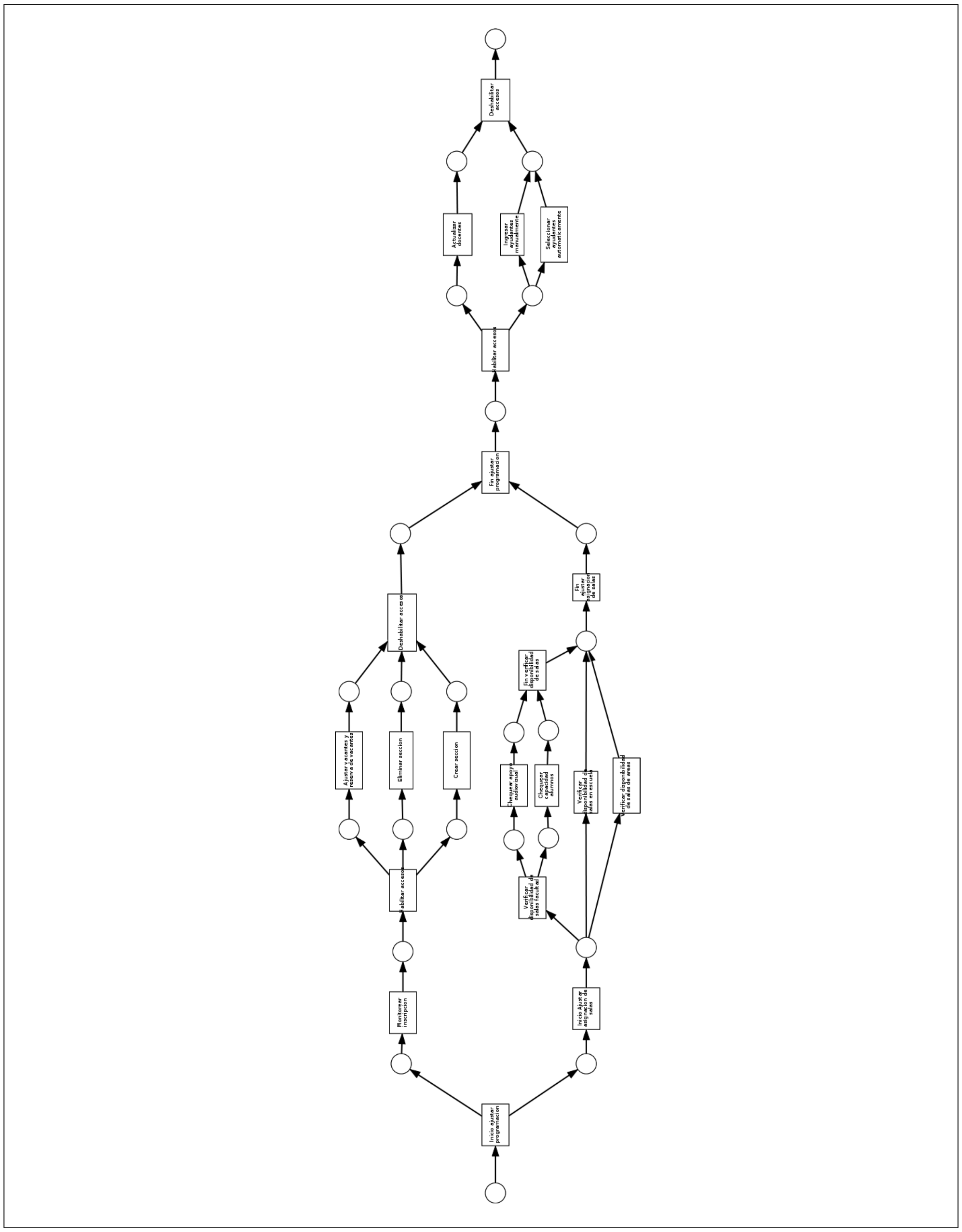

Inicio ajustar programacion
Monitorear inscripcion
Habilitar accesos
Ajustar vacantes y reserva de vacantes
Eliminar seccion
Crear seccion
Deshabilitar accesos
Inicio Ajustar asignacion de salas
Verificar disponibilidad de salas facultad
Chequear apoyo audiovisual
Chequear capacidad alumnos
Fin verificar disponibilidad de salas
Verificar disponibilidad de salas en escuela
Verificar disponibilidad de salas de areas
Fin ajustar asignacion de salas
Fin ajustar programacion
Habilitar accesos
Actualizar docentes
Ingresar ayudantes manualmente
Seleccionar ayudantes automaticamente
Deshabilitar accesos

FIGURE A.4. Adjust Academic Program sub-model

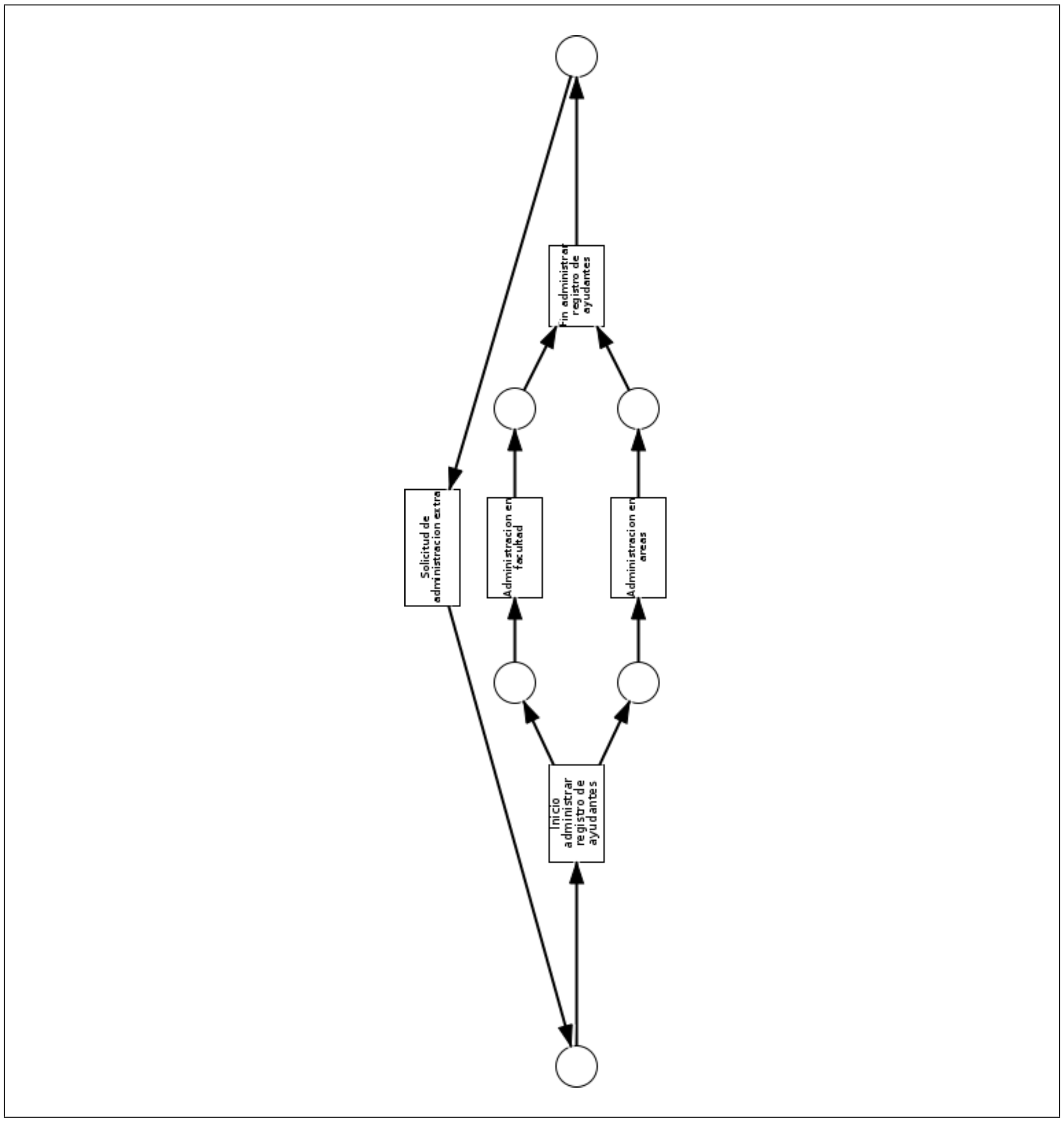

Fin administrar registro de ayudantes
Solicitud de administracion extra
Administracion en facultad
Administracion en areas
Inicio administrar registro de ayudantes

FIGURE A.5. Student Assistant Planning sub-model

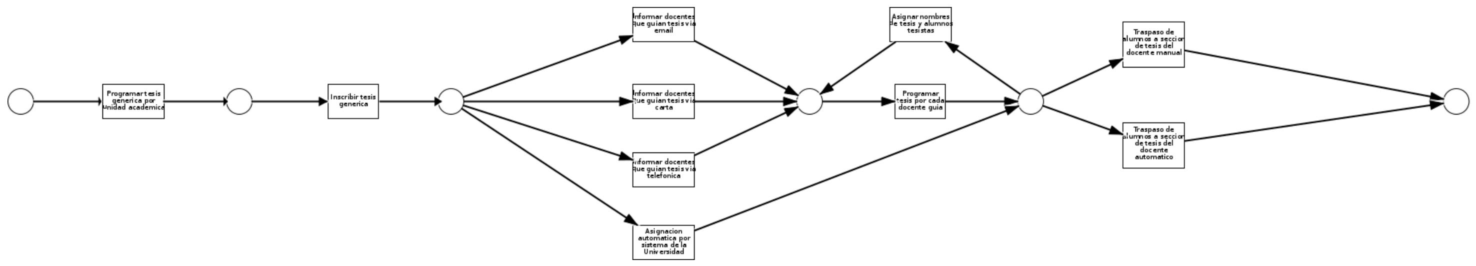

Programar tesis generica por unidad academica
Inscribir tesis generica
Informar docentes que guian tesis via email
Informar docentes que guian tesis via carta
Informar docentes que guian tesis via telefonica
Asignacion automatica por sistema de la Universidad
Asignar nombres de tesis y alumnos tesistas
Programar tesis por cada docente guia
Traspaso de alumnos a seccion de tesis del docente manual
Traspaso de alumnos a seccion de tesis del docente automatico

FIGURE A.6. Thesis Planning sub-model