

Article

# Strategies to Automatically Derive A Process Model From A Configurable Process Model Based on Event Data

Mauricio Arriagada-Benítez<sup>1,\*</sup>, Marcos Sepúlveda<sup>1</sup>, Jorge Muñoz-Gama<sup>1</sup>, and Joos C. A. M. Buijs<sup>2</sup>

<sup>1</sup> Computer Science Department, School of Engineering, Pontificia Universidad Católica de Chile, Vicuña Mackenna 4860, Santiago (Chile), Tel.: +56 2 2354 4950; marcos@ing.puc.cl; jmun@uc.cl

<sup>2</sup> Eindhoven University of Technology, Groene Loper 5, Eindhoven (The Netherlands); j.c.a.m.buijs@tue.nl

\* Correspondence: mauricio.arriagada@uc.cl; Tel.: +56 2 2354 1795

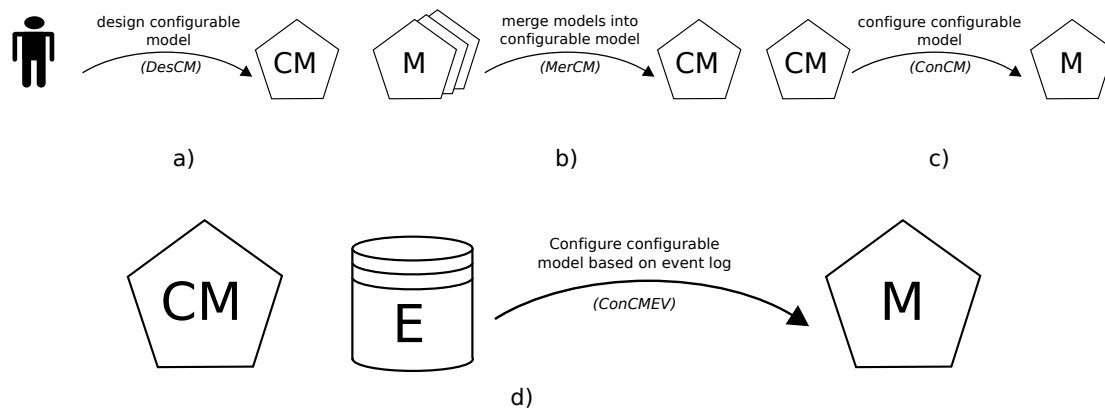
**Abstract:** Configurable process models are frequently used to represent business workflows and other discrete event systems among different branches of large organizations: they unify commonalities shared by all branches and describe their differences, at the same time. The configuration of such models is usually done manually, which is challenging. 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 at the same time, since choices influence each other. Nowadays, information systems that support the execution of business processes create event data reflecting how processes are performed. In this article, we propose three strategies (based on exhaustive search, genetic algorithms, and greedy heuristic) that use event data to automatically derive a process model from a configurable process model that better represents the characteristics of the process in a specific branch. These strategies have been implemented in our proposed framework, and tested in both business-like event logs as recorded in a higher educational ERP system, and a real case scenario involving a set of Dutch municipalities.

**Keywords:** business workflows; discrete event systems; event logs; configurable process models; configurable process trees; process mining; business processes

## 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 [1]. 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 products and services, but the processes in the back-office differ significantly. 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* [2], *C-BPEL* [3], *C-EPC* [4], *C-PT* [5], and *C-BPMN* [6].

The importance of having a configurable process model to describe process variants has been a subject of study in the literature. In [7] 20 Business Process Management (BPM) use cases are identified. Three of those use cases involve configurable process models: *design configurable model* (*DesCM*), *merge models into configurable model* (*MerCM*), and *configure configurable model* (*ConCM*) (see Figure 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, the original use case does 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 [8]. More specifically, process discovery techniques aim at creating a process model based on the historical behavior recorded in an event log. Even though some process mining techniques have been applied to support the creation and derivation of configurable process models (e.g., [9]), the approach was simplistic and not executable on real-life data.

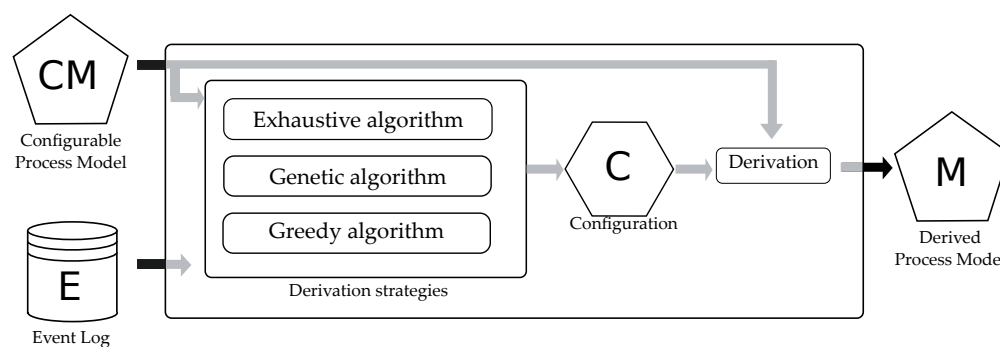


**Figure 1.** BPM use cases [7] related to configurable process models: a) manually design a configurable process model, b) merge collection of process models to generate a configurable process model, and c) configure a configurable process model to obtain a process model. In this article, we propose d) as a new use case: to derive a process model based on a configurable process model and an event log.

While analyzing the literature about configurable process models, we have identified two main challenges. 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. At the same time, there is a growing availability of event data in organizations, which also pressures for having specialized, smart and efficient techniques to include historical data for configuring a configurable process model. Therefore, 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 could be defined in a local context. So, a second challenge is to create a simple representation for the different degrees of freedom we want to allow in the configurable nodes [10].

This article presents an approach with a threefold objective. First, we propose an additional use case to those presented by [7], where we combine a configurable process model (manually or automatically generated [9]) and an event log to derive a process model that better represents the observed behavior in the historical data, depicted in Figure 1 as *ConCMEV*. Second, to support this use case extension, we redefine the configurable process model representation, in particular how to represent configurable process trees [10], 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 2](#), both inputs are part of the extended configurable process model use case. The 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 2.** Overview of the proposed framework to derive a process model. Three alternative derivation strategies allow to obtain a configuration that later on is used to derive a process model from the configurable process model.

This article is organized as follows: related work is described in [section 2](#) and then [section 3](#) introduces the theoretical foundation of the proposed framework. In [section 4](#), we present the methodology that describes 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. Results and discussions are presented in [section 5](#). Finally, conclusions and future work are presented in [section 6](#).

## 2. 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 obtaining such a configurable process model [10–15]. Manual (i.e. by the user) process model configuration has been addressed by [10,16]. In [16], 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. [10] uses a configurable tree-like representation which is sound by construction. Applied in the *CoSeLoG* project [17], this approach merges variants of different municipalities to create a configurable process model. The same author underlines in [5] 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 KPI satisfaction. The results are then presented to the end user, who then inspects the proposed configurations and selects one to be applied.

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 [18], which is able to discover a process model including configurations, when given multiple event logs. However, the performance of the

Evolutionary Tree Miner is not optimal. Because it is an evolutionary algorithm, and the configuration aspect adds many possibilities, the challenge of discovering a process model and a configuration becomes challenging.

The main limitation of existing approaches is the restricted number of configurations a configurable process model can have. The approach proposed in this paper aims to enhance the work already done in configurable process models by allowing a large number of configurations and by providing 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. Similar to existing approaches, we implemented our techniques in the *ProM* process mining framework.

### 3. Theoretical Foundation of the Framework for Deriving a Process Model From a Configurable Process Model

In this section, 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.

#### 3.1. Preliminaries - 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,  $b_1 = []$ ,  $b_2 = [x, x, y]$ ,  $b_3 = [x, y, z]$ ,  $b_4 = [x, x, y, x, y, z]$ ,  $b_5 = [x, y^3, z]$  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  $n$ ,  $\rho = \langle a_1, a_2, a_3, \dots, a_n \rangle \in A^*$ , is a mapping  $\{1, \dots, n\} \rightarrow A$ . Its length is denoted by  $|\rho| = n$  and the element at position  $i$  ( $a_i$ ) is denoted as  $\rho_i$ . Also,  $\langle \rangle$  is the empty sequence. Note that sequences are written using angle brackets. For two sequences,  $\rho_1$  and  $\rho_2$ ,  $\rho_1 \cdot \rho_2$  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 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 2** (Projection). Let  $A$  be a set and  $A' \subseteq A$  one of its subsets.  $\sigma \upharpoonright_{A'}$  denotes the projection of  $\sigma \in A^*$  on  $A'$ , e.g.,  $\langle a, a, b, c \rangle \upharpoonright_{\{a,c\}} = \langle a, a, c \rangle$ . The projection can also be applied to multisets, e.g.,  $[x^3, y, z^2] \upharpoonright_{\{x,y\}} = [x^3, y]$ .

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

### 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 [8]. The use of Petri nets as modeling notation is common in both Discrete Event Systems and Process Mining literature [8]. 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 [18]. A *process tree* [18,19] 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 ( $\odot$ ). A silent activity is denoted by  $\tau$  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 [20], Inductive Miner [21] and Inductive Miner-infrequent [22]. Its formal definition is as follows:

**Definition 3** (Process tree). Let  $A$  be a finite set of activities, with  $\tau \notin A$  representing a silent activity.  $\oplus = \{\rightarrow, \times, \vee, \wedge, \odot\}$  is the set of *process tree operators*. A *process tree* is recursively defined as follows [8]:

- if  $a \in A \cup \{\tau\}$ , then  $Q = a$  is a process tree,
- if  $Q_1, Q_2, \dots, Q_n$  are process trees where  $n \geq 1$ , and  $\oplus \in \{\rightarrow, \times, \vee, \wedge\}$ , then  $Q = \oplus(Q_1, Q_2, \dots, Q_n)$  is a process tree, and
- if  $Q_1, Q_2, \dots, Q_n$  are process trees where  $n \geq 2$ , then  $Q = \odot(Q_1, Q_2, \dots, Q_n)$  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, [8] 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 [8]. However, process trees also preserve interesting properties for the analysis and the verification, such as soundness by construction [8], and the block-structured [8].

Figure 3 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 ( $\odot$ ). 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 ( $\odot$ ) 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  $g$ .

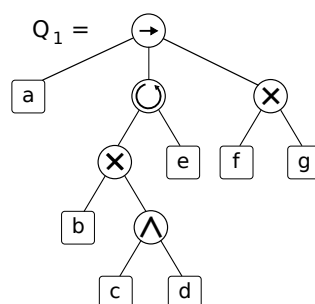


Figure 3. Example of a process tree model.

### 3.4. Configurable Process Tree

Representing configurable process models as process trees has been addressed by [9,10]. 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 [23]. 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.

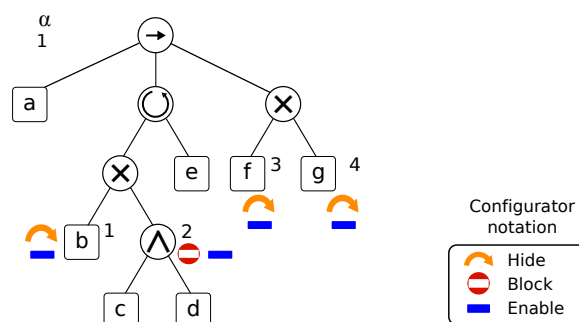
Enable (allow) means the node is enabled to be visited, hide means the node is to be skipped over, and block means the node cannot be reached. The foundation of configurable process trees is described in [10], and also applied in [9].

**Definition 4** (Process tree configurators).  $\Theta = \{H, B, E\}$  is the set of *process tree configurators*, and  $\Theta_{\times} = \{\{H\}, \{B\}, \{E\}, \{H, B\}, \{H, E\}, \{B, E\}, \{H, B, E\}\}$  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  $\tau$  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 [18].
- *E*: *enable* a node. It essentially allows a node to be performed, either an operator or an activity, so that it behaves normally.

**Definition 5** (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*  $\alpha : N(Q) \rightarrow \Theta_{\times}$  defining a configuration set for some nodes.  $N(Q) \stackrel{\alpha}{\subseteq} N(Q)$  is the set of *configurable nodes*, i.e.,  $N^{\alpha}(Q^{\alpha}) = \text{domain}(\alpha)$ . For the sake of clarity, let us assume an ordering among the configurable nodes, i.e.,  $n_1, n_2, \dots, n_{|N^{\alpha}(Q^{\alpha})|}$ .  $C(Q^{\alpha})$  is the set of all possible *configurations* of  $Q^{\alpha}$ , i.e.,  $C(Q^{\alpha}) = \{ \langle c_1, c_2, \dots, c_{|N^{\alpha}(Q^{\alpha})|} \rangle \mid c_i \in \alpha(n_i) \}$ .

Figure 4 is an example of a configurable process tree  $Q_1^{\alpha}$  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*.



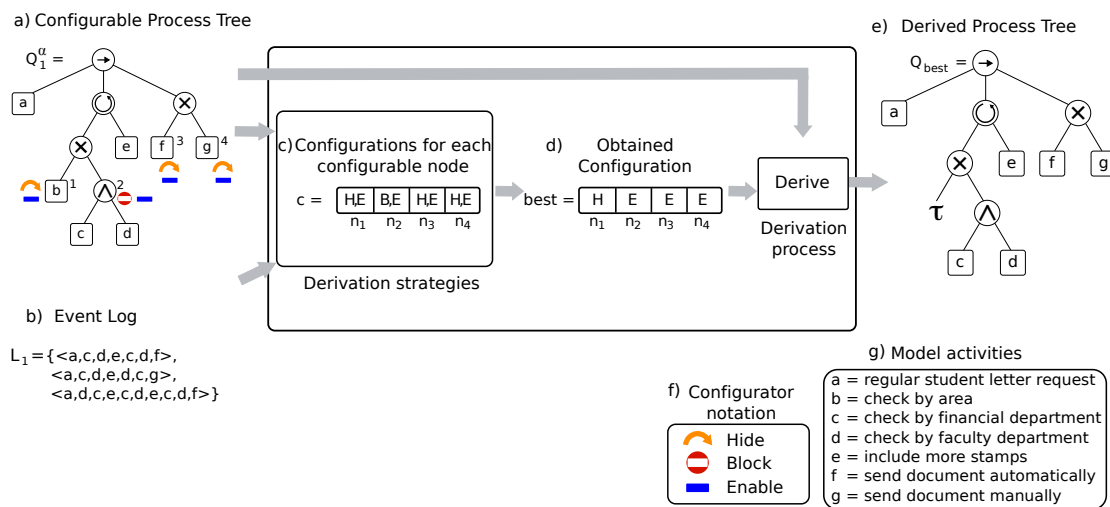
**Figure 4.** Example of a configurable process tree containing 4 configurable nodes.

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 6** (Derived process tree). Let  $Q^\alpha$  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  $Q^\alpha$  by applying the configuration  $c$ , using the rules applied in [18].

Figure 5 presents a running example of the execution of the derivation framework shown in Figure 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 5 also depicts the notation for the process tree configurators and the model activities in f) and g), respectively.



**Figure 5.** Running example that illustrates the derivation framework. A configurable process tree and event log, shown in a) and b), are the inputs. The internal representation of a configuration used by the three strategies is represented in c). The obtained configuration is shown in d). Finally, the output, a derived process tree is shown in e).

### 3.5. Quality of a Derived Process Tree

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* [8,20,24]. 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 [24].

**Definition 7** (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 [18] 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:

$$conformance(Q, L, W) = \frac{f(Q, L) \cdot w_f + p(Q, L) \cdot w_p + g(Q, L) \cdot w_g + s(Q, L) \cdot w_s}{w_f + w_p + w_g + w_s} \quad (1)$$

In this article, we have defined the conformance metric based on [18] and used the corresponding implementation to evaluate the quality of a given process tree. 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 represents a given event log, e.g. has the best conformance value.

**Definition 8** (Optimal configuration). Let  $Q^\alpha$  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. Methodology

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 [18]. In [18], the input is a collection of event logs and the output is a configurable process model. In our case, the input is 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.

##### 4.1. Obtaining a configuration based on the exhaustive strategy

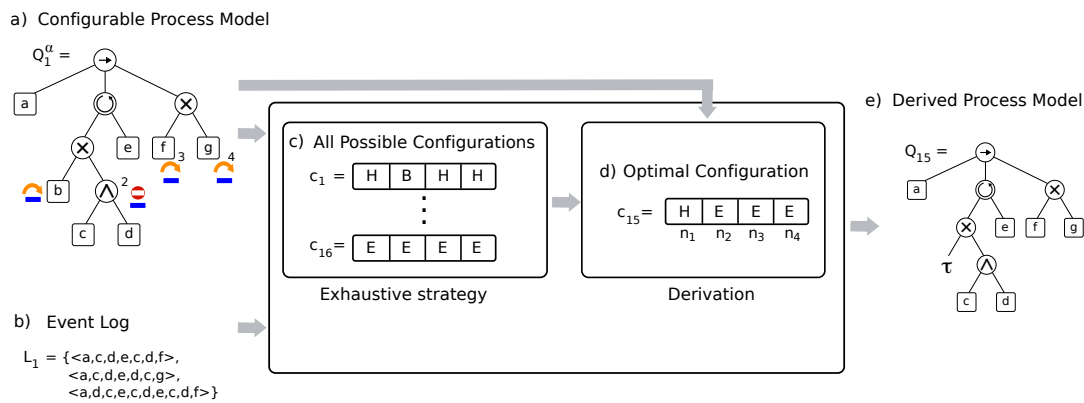
The first strategy in the proposed framework is the exhaustive strategy, whose relevance is that it ensures obtaining the best configuration among all possible ones. In general, an exhaustive strategy is a brute-force method to a problem involving the search for a solution among all possible ones, e.g., those obtained from combinatorial objects, such as permutations or combinations. In this case, for a configurable process tree  $Q^\alpha$ , 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  $derive(Q^\alpha, c)$  is used to obtain a derived model  $m$  from the CPT  $Q^\alpha$ , given the configuration  $c$ .
- The model  $m$  is evaluated using the *conformance* function defined in Equation 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 6 presents an illustrative example of the exhaustive strategy. Given the CPT  $Q_1^\alpha$  and the log  $L_1$ , shown in a) and b), the framework finds the best configuration and the corresponding derived process tree. The CPT  $Q_1^\alpha$ , 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  $Q_1^\alpha$  to derive the process tree  $Q_{15}$ , shown in e).



**Figure 6.** Overview of the exhaustive strategy to derive a process tree. The algorithm generates all possible configurations, in order to find an optimal configuration. This configuration is used to obtain an optimal derived 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:    $C(Q^\alpha) \leftarrow CartesianProduct(\alpha(n_1), \alpha(n_2), \dots, \alpha(n_{|N^\alpha(Q^\alpha)|}))$ 
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:     end if
9:   end for
10:  return  $best$ 
11: end procedure

```

---

#### 4.2. 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 [25–27]. 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 7 illustrates these main elements.

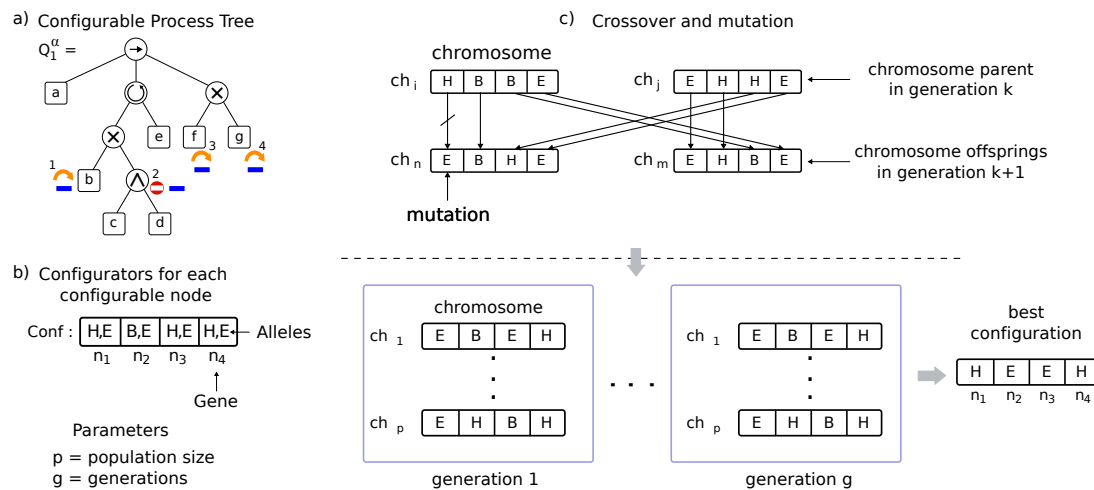
- **Representation** : In GA, a *chromosome* represents a potential solution and it is formed by a *genes* chain. Genes represent distinct aspects of the solution as a whole, just as human genes represent distinct aspects of individual people, such as their gender or eye color. A potential

value of a gene is called an *allele*. In our case, a chromosome represents a configuration  $c \in C(Q^\alpha)$ , see Figure 7 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  $i \in 1, \dots, |N^\alpha(Q^\alpha)|$ .

- **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 [28].
- **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<sup>1</sup> that evaluates the quality of a *chromosome* to either be selected for the next generation or to be discarded. We refer to [25] 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 7 c). Given two chromosomes  $a$  and  $b$  and a cutting point  $1 \leq i < |N^\alpha(Q^\alpha)|$ , the offspring chromosomes are  $\langle a_1, \dots, a_i, b_{i+1}, \dots, b_{|N^\alpha(Q^\alpha)|} \rangle$  and  $\langle b_1, \dots, b_i, a_{i+1}, \dots, a_{|N^\alpha(Q^\alpha)|} \rangle$ . 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 subsection 3.4.
- **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 [25].

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 [18,25] for more details.

<sup>1</sup> 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 subsection 3.5, but to the conformance function. Therefore and for the sake of clarity, we refer to it as conformance function.



**Figure 7.** Overview of the genetic strategy to derive a process tree. The internal configuration representation allows this evolutionary algorithm to use crossover and mutation operations to generate new candidate configurations to be assessed.

---

#### Algorithm 2 Obtain a configuration based on the genetic strategy

---

```

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

```

---

#### 4.3. Obtaining a configuration based on the greedy strategy

The above described evolutionary strategy is able to find a good configuration (potentially an optimal one [25]) 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 [29]. 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 subtrees 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  $\odot$  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 8 (a); it can have only *dependent* configurable nodes, as shown in Figure 8 (b); or it can combine both *independent* and *dependent* configurable nodes, as shown in Figure 8 (c).

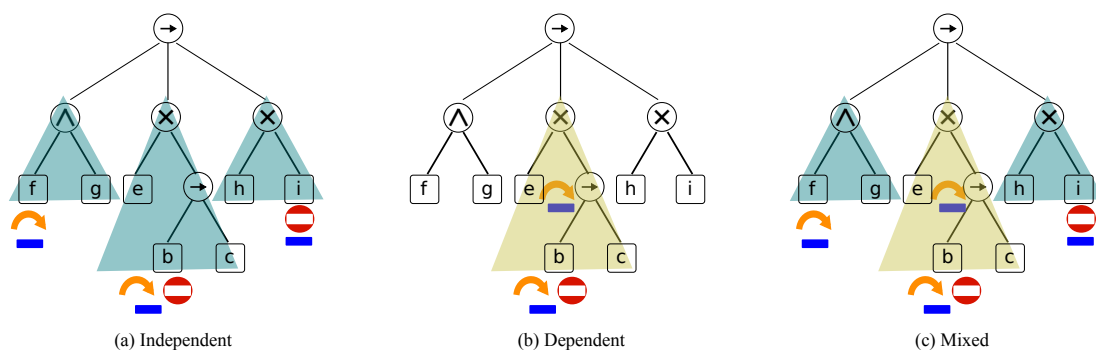


Figure 8. Different configurable node dependencies.

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 ancestor that is a loop operator, then the new root is the loop operator ancestor that is close to the original root. Such a new subtree might contain other pending configurable nodes; they are temporarily set to  $\tau$  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 9 illustrates the greedy strategy to find a configuration for the configurable process tree  $Q_1^\alpha$ . The derivation process is shown on the right 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  $\odot$  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  $\tau$  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  $\tau$  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  $Q_1^\alpha$ , the projection of the activities contained 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  $Q_1^\alpha$  and  $L_1$ , and  $Q_2$  is the derived process tree from  $Q_1^\alpha$  and  $L_2$ .

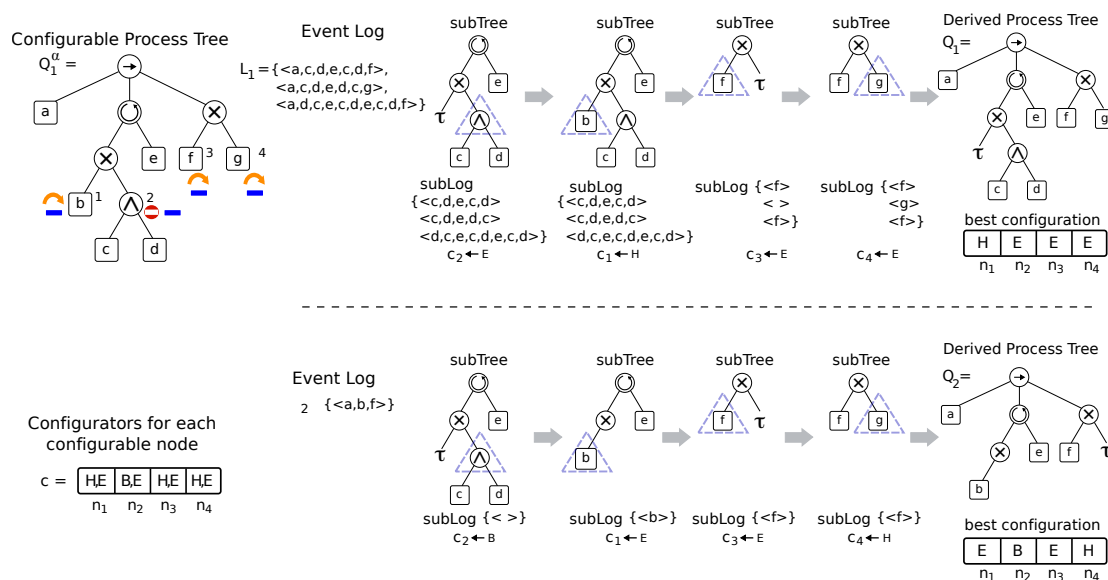


Figure 9. Overview of the greedy strategy to derive a process tree. There are two event log inputs to show different subtrees and sublogs scenarios to finally derive a process tree.

**Algorithm 3** Obtaining a configuration based on the greedy strategy

---

```

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

```

---

## 5. Result and Discussion

The proposed strategies have been implemented and evaluated in two different scenarios. In this section, we first describe how the framework has been implemented (see [subsection 5.1](#)). Then, we describe the two scenarios considered: a realistic scenario based on the adoption of a higher educational ERP system by some universities (see [subsection 5.2](#)), and a scenario that represents a real-life registration process [30], which is executed on a daily basis by Dutch municipalities (see [subsection 5.3](#)). Afterwards, in [subsection 5.4](#), we analyze the performance of the proposed strategies on a set of controlled experiments created for testing the performance of the strategies depending on both the log complexity and the model complexity.



### 5.1. Implementation

We have implemented the three strategies, exhaustive, genetic and greedy, as three plug-ins of the ProM process mining framework<sup>2</sup>, within the *ConfigurableProcesses* package. 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.

### 5.2. 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 article 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.

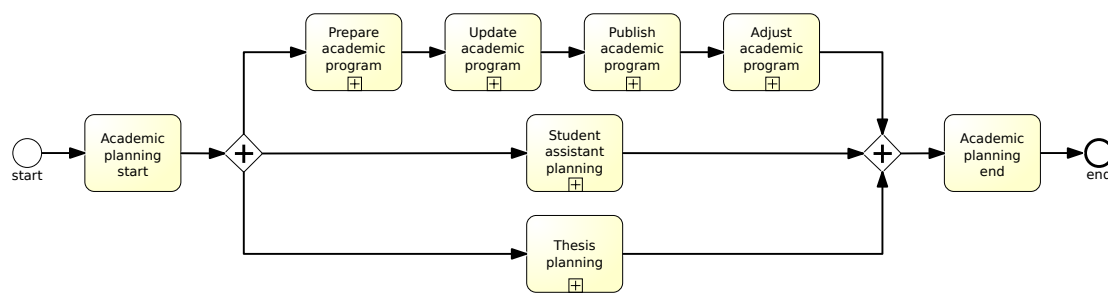
#### *Academic planning process*

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 10](#) depicts a general process model that includes sub-processes that group common activities in the academic planning process.

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.

---

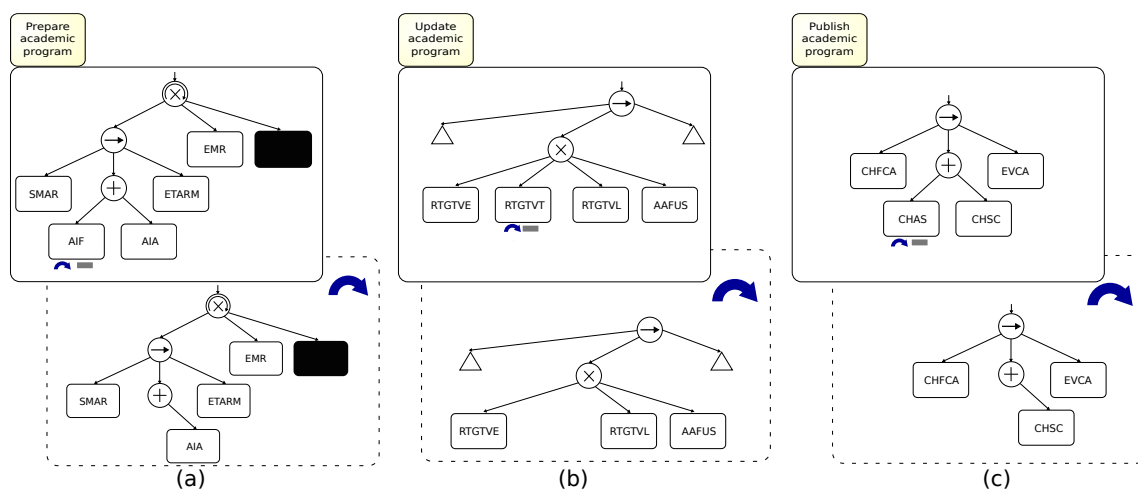
<sup>2</sup> Available in the ProM nightly-builds <http://www.promtools.org/prom6/nightly>



**Figure 10.** General model of the academic planning process of a university. It includes parallel flows that contain sub-processes.

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, grouped in the 8 sub-processes shown in [Figure 10](#), and 9 configurable nodes that have *mixed* dependencies. The configurable process model is considerably large, making it impractical to show it completely. To illustrate how the configuration is performed, [Figure 11](#) shows a scheme of the configuration process. The upper boxes represent 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.



**Figure 11.** General idea of process model derivation for three different configurable sub-processes.

Having a reference process model, whose overall view is shown in [Figure 10](#), 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 cases. The results are shown in [Table 1](#), including fitness, precision, and the conformance metric, calculated as 90% fitness and 10% precision.

It is possible to observe in [Table 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

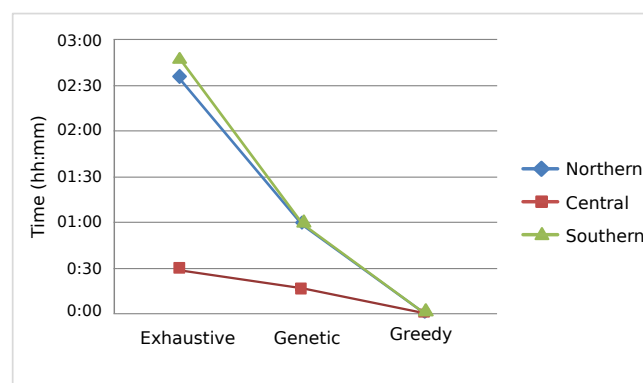
in the reference model. A decision maker would probably decide the ERP system that is desired to acquire is not suitable for this university.

**Table 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	<b>0.922</b>	0.986	0.510	<b>0.939</b>	0.690	0.510	<b>0.672</b>
Genetic	0.981	0.398	<b>0.922</b>	0.986	0.510	<b>0.939</b>	0.690	0.510	<b>0.672</b>
Greedy	0.981	0.370	<b>0.920</b>	0.986	0.500	<b>0.938</b>	0.690	0.500	<b>0.671</b>

Table 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 12. It is possible to observe that the time required by the *exhaustive* strategy is 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 [31].



**Figure 12.** 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.

### 5.3. Real-life experiments

In order to evaluate the performance of the techniques on real-life data, we apply all three strategies on a real-life dataset. The data used contains event data from the building permits process of five Dutch municipalities [17,32]. 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) [18, 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.

We use CPTs based on the models discovered by the ETM on this dataset, which are shown in Figure 13. Figure 14 shows the two CPTs created based on the model shown in Figure 13a. The CPT in Figure 14a has two configurable nodes while the CPT in Figure 14b has eight configurable nodes, all randomly chosen. Notice that we could not create a CPT similar to Figure 13a because we are not considering the downgrade operator used in [18]. Figure 15 shows the three CPTs created based on the model shown in Figure 13b. The CPT in Figure 15a is equivalent to the one shown in Figure 13b. The CPT in Figure 15b has six configurable nodes while the model in Figure 15c 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 models 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.

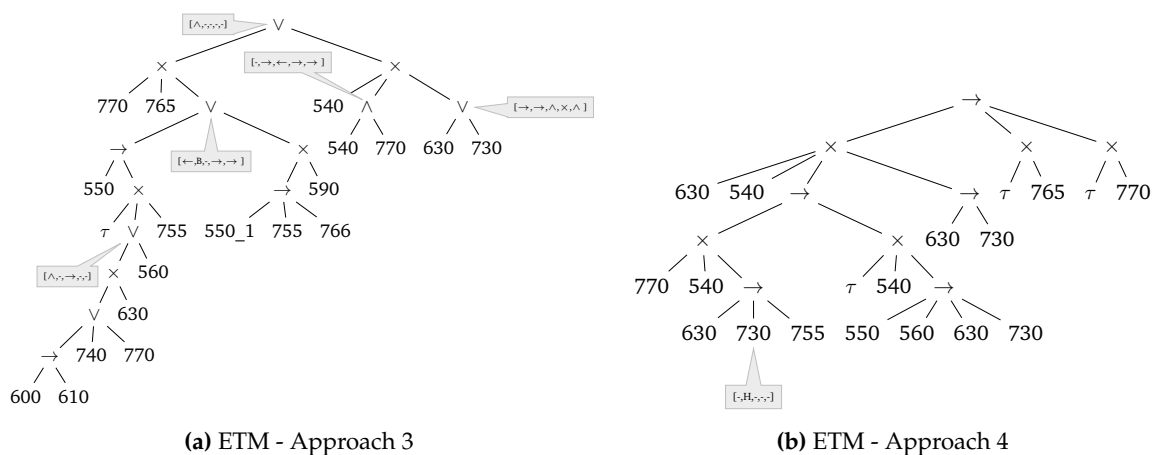


Figure 13. Process trees originally discovered by the ETM on the WABO event logs.

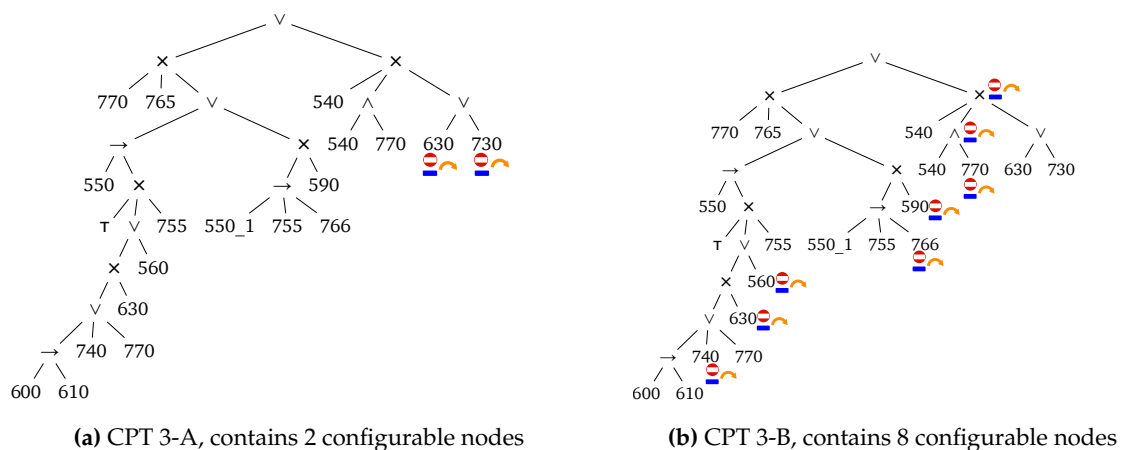
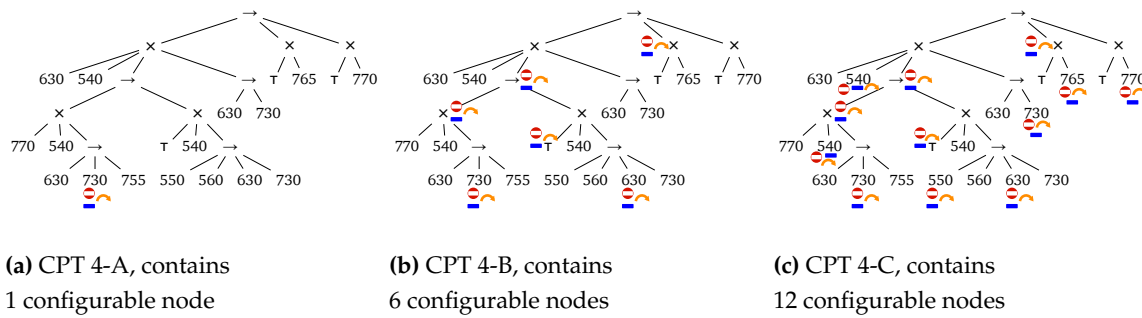


Figure 14. CPTs based on the model discovered by the ETM Approach 3.

The results of the experiments with the CPTs created based on the model discovered by the ETM Approach 3 are shown in Table 2 and Table 3. Table 4, Table 5 and Table 6 show the results of the experiments with the CPTs created based on the model discovered by the ETM Approach 4. All tables include fitness, precision, and the conformance metric, calculated as 90% fitness and 10% precision.



**Figure 15.** CPTs based on the model discovered by the ETM Approach 4.

**Table 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	<b>0.921</b>	0.962	0.768	<b>0.943</b>	0.936	0.646	<b>0.907</b>	0.985	0.751	<b>0.962</b>	0.950	0.797	<b>0.935</b>
Genetic	0.941	0.745	<b>0.921</b>	0.962	0.768	<b>0.943</b>	0.936	0.646	<b>0.907</b>	0.985	0.751	<b>0.962</b>	0.950	0.797	<b>0.935</b>
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	<b>0.937</b>	0.948	0.979	<b>0.951</b>	0.927	0.729	<b>0.907</b>	0.984	0.917	<b>0.977</b>	0.957	0.909	<b>0.952</b>

f = fitness, p = precision, fpc = conformance

**Table 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	0.941	0.745	0.921	0.962	0.768	0.943	0.936	0.646	<b>0.907</b>	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	<b>0.907</b>	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	<b>0.937</b>	0.948	0.979	<b>0.951</b>	0.927	0.729	<b>0.907</b>	0.984	0.917	<b>0.977</b>	0.957	0.909	<b>0.952</b>

f = fitness, p = precision, fpc = conformance

**Table 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	0.958	0.730	0.935	0.962	0.912	0.957	0.945	0.648	<b>0.915</b>	0.993	0.849	<b>0.979</b>	0.954	0.931	<b>0.952</b>
Genetic	0.958	0.730	0.935	0.962	0.912	0.957	0.945	0.648	<b>0.915</b>	0.993	0.849	<b>0.979</b>	0.954	0.931	<b>0.952</b>
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	<b>0.937</b>	0.948	0.979	<b>0.951</b>	0.927	0.729	0.907	0.984	0.917	0.977	0.957	0.909	<b>0.952</b>

f = fitness, p = precision, fpc = conformance

**Table 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	<b>0.915</b>	0.980	0.962	<b>0.978</b>	0.948	0.894	<b>0.943</b>	0.986	0.978	<b>0.985</b>	0.949	0.975	<b>0.952</b>
Genetic	0.911	0.951	<b>0.915</b>	0.980	0.962	<b>0.978</b>	0.948	0.894	<b>0.943</b>	0.986	0.978	<b>0.985</b>	0.949	0.975	<b>0.952</b>
Greedy	0.911	0.951	<b>0.915</b>	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	<b>0.978</b>	0.948	0.894	<b>0.943</b>	0.986	0.978	<b>0.985</b>	0.949	0.975	<b>0.952</b>

f = fitness, p = precision, fpc = conformance

It can be observed that in all cases the different strategies obtain good results. 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.

**Table 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	<b>0.919</b>	0.980	0.962	<b>0.978</b>	0.957	0.894	<b>0.951</b>	0.986	0.978	<b>0.985</b>	0.949	0.975	<b>0.952</b>
Genetic	0.915	0.951	<b>0.919</b>	0.980	0.962	<b>0.978</b>	0.957	0.894	<b>0.951</b>	0.986	0.978	<b>0.985</b>	0.949	0.975	<b>0.952</b>
Greedy	0.915	0.951	<b>0.919</b>	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	<b>0.978</b>	0.948	0.894	0.943	0.986	0.978	<b>0.985</b>	0.949	0.975	<b>0.952</b>

f = fitness, p = precision, fpc = conformance

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 4 corresponding to CPT 4-A, shown in Figure 15a), 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 15b and Figure 15c, respectively), the search space is larger, so eventually there could be a better configuration. This actually occurs in Variant 1 in Table 5, and Variant 1 and Variant 3 in Table 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 5 and Table 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 14a and Figure 14b, 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 2 and Table 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).

#### 5.4. 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*.

##### 5.4.1. 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 will be 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.

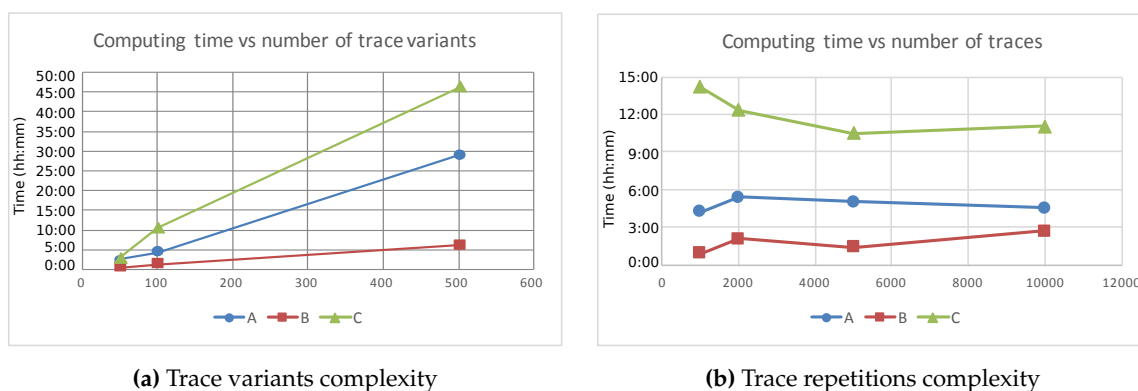
Table 7 summarizes the experiments. In the first set of experiments, the number of trace variants is varied. In this case, a random number of trace repetitions between 10 and 20 is considered for each trace variant. The final number of traces is therefore different for each university. In the second 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 8 displays the results obtained with each algorithm when varying the log complexity.

**Table 7.** Different log settings for three universities

	#trace variants	#trace repetitions	#traces in the event log for a target model A	#traces in the event log for a target model B	#traces 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

### Exhaustive strategy evaluation

As a force-brute method, this strategy searches over all possible configurations that can be applied to the configurable process model. As seen in Figure 16a, 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 16b 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.



**Figure 16.** Performance of the exhaustive strategy when varying log complexity.

### 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 17a shows there is a linear dependency between the computing time of the *genetic* strategy and the number of trace variants. Meanwhile, Figure 17b 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.

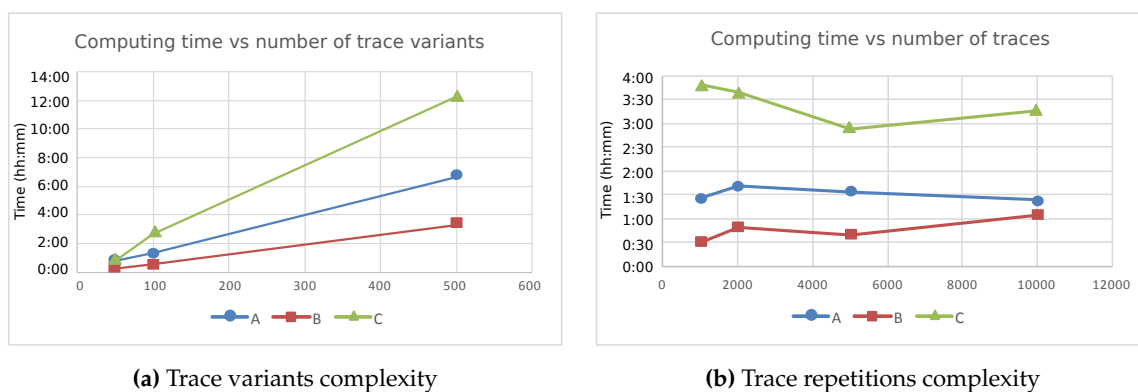
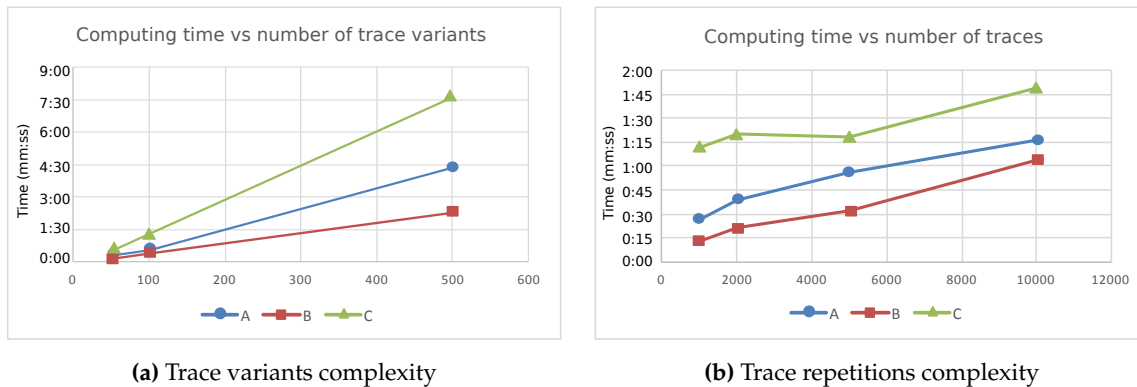


Figure 17. Performance of the genetic strategy when varying log complexity.

### 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 18a 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 18b illustrates a slight ascending computing time when increasing the number of trace repetitions while keeping the number of trace variants constant.

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 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 strategy 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.



**Figure 18.** Performance of the greedy strategy when varying the log complexity.

**Table 8.** Results obtained for all algorithms considering different log complexity.

	Log complexity	A		B		C	
		time (hh:mm:ss)	conformance	time (hh:mm:ss)	conformance	time (hh:mm:ss)	conformance
exhaustive	50t 10-20rep	2:28:23	0.922	0:27:36	0.939	2:41:04	0.672
	100t 10-20rep	4:08:03	0.923	1:10:33	0.935	10:04:35	0.784
	500t 10-20rep	27:40:27	0.924	5:55:05	0.941	44:14:38	0.787
	100t 10rep	4:04:22	0.922	0:50:08	0.939	13:36:03	0.782
	100t 20rep	5:10:53	0.921	2:00:30	0.939	11:49:07	0.782
	100t 50rep	4:50:08	0.924	1:20:34	0.939	10:04:29	0.788
	100t 100rep	4:21:51	0.924	2:32:39	0.935	10:36:01	0.784
	genetic	50t 10-20rep	0:56:10	0.922	0:16:08	0.939	0:56:43
100t 10-20rep		1:34:54	0.923	0:41:19	0.935	3:13:04	0.784
500t 10-20rep		7:57:34	0.924	3:56:26	0.941	14:40:04	0.787
100t 10rep		1:21:56	0.922	0:28:26	0.939	3:39:13	0.781
100t 20rep		1:37:05	0.921	0:47:12	0.939	3:30:46	0.782
100t 50rep		1:30:28	0.924	0:37:23	0.939	2:46:14	0.788
100t 100rep		1:20:15	0.924	1:01:52	0.935	3:07:46	0.784
greedy		50t 10-20rep	0:00:18	0.92	0:00:08	0.938	0:00:29
	100t 10-20rep	0:00:32	0.92	0:00:23	0.933	0:01:16	0.784
	500t 10-20rep	0:04:10	0.922	0:02:11	0.94	0:07:19	0.786
	100t 10rep	0:00:30	0.92	0:00:15	0.938	0:01:22	0.78
	100t 20rep	0:00:45	0.919	0:00:25	0.937	0:01:32	0.782
	100t 50rep	0:01:05	0.921	0:00:37	0.938	0:01:30	0.787
	100t 100rep	0:01:28	0.922	0:01:13	0.934	0:02:06	0.784

#### 5.4.2. 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  $\cup$  operator nodes, and models that consider all operator nodes ( $\times$ ,  $\wedge$  and  $\cup$ ). 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 subsection 4.3). In total, there are 9 different configurable process models  $Q_i^a$ , where  $i = 1, \dots, 9$ ; they are summarized in Table 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 9, they are represented as  $L_{i_1}$ ,  $L_{i_2}$  and  $L_{i_3}$ , for  $i = 1, \dots, 9$ .

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 9, including the computing time and the conformance metric. The computing time required for all strategies is shown in Figure 19, where the average computing time required for the 3 event logs created for each configurable process model is displayed.

Figure 19a 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  $\circ$  operators. This is due to the conformance method requires more time to handle parallelism, because the traces to be processed are usually longer [24]. Whereas models with a topology that only contains  $\times$  and  $\circ$  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 19a for the *genetic* strategy and in Figure 19a 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 19a also depicts that the *greedy* strategy to derive a process model, requires considerable less computing time than the other two strategies.

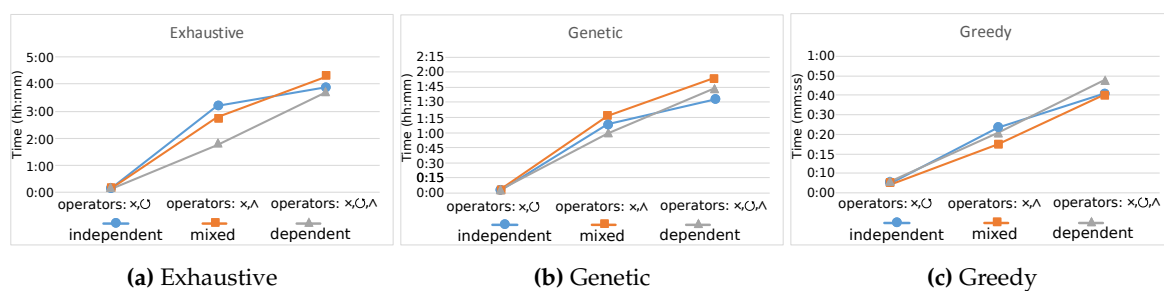


Figure 19. Performance of the different strategies when varying the model complexity.

## 6. Conclusions

In this paper, 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. Several related works can be performed to extend this research. First of all, user-defined rules can be used to guide the configuration process [5]. 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

**Table 9.** Results obtained for all strategies considering different model complexity.

Configurable model topology	Configurable node dependencies	Event log	Strategies					
			Exhaustive		Genetic		Greedy	
			time (hh:mm:ss)	conformance	time (hh:mm:ss)	conformance	time (hh:mm:ss)	conformance
Q <sub>1</sub> <sup>a</sup>	Independent	L <sub>1,1</sub>	00:08:15	0,98	00:01:46	0,98	00:00:04	0,98
		L <sub>1,2</sub>	00:06:43	0,99	00:01:03	0,99	00:00:02	0,98
		L <sub>1,3</sub>	00:12:47	0,84	00:02:42	0,84	00:00:05	0,84
Q <sub>2</sub> <sup>a</sup>	Mixed	L <sub>2,1</sub>	00:07:05	0,98	00:01:35	0,98	00:00:03	0,97
		L <sub>2,2</sub>	00:05:25	0,98	00:01:19	0,98	00:00:02	0,97
		L <sub>2,3</sub>	00:09:17	0,84	00:02:49	0,84	00:00:05	0,83
Q <sub>3</sub> <sup>a</sup>	Dependent	L <sub>3,1</sub>	00:07:51	0,98	00:02:15	0,98	00:00:05	0,98
		L <sub>3,2</sub>	00:06:20	0,98	00:01:56	0,98	00:00:04	0,98
		L <sub>3,3</sub>	00:08:18	0,89	00:03:37	0,89	00:00:06	0,87
Q <sub>4</sub> <sup>a</sup>	Independent	L <sub>4,1</sub>	03:47:51	0,93	01:01:16	0,93	00:00:25	0,93
		L <sub>4,2</sub>	02:54:15	0,95	00:45:41	0,95	00:00:14	0,94
		L <sub>4,3</sub>	04:49:52	0,87	01:29:55	0,87	00:00:47	0,87
Q <sub>5</sub> <sup>a</sup>	Mixed	L <sub>5,1</sub>	02:43:11	0,95	01:06:10	0,95	00:00:19	0,95
		L <sub>5,2</sub>	01:57:22	0,95	00:48:31	0,95	00:00:09	0,95
		L <sub>5,3</sub>	05:18:59	0,89	01:44:15	0,89	00:00:36	0,89
Q <sub>6</sub> <sup>a</sup>	Dependent	L <sub>6,1</sub>	01:59:09	0,94	00:49:10	0,94	00:00:24	0,94
		L <sub>6,2</sub>	01:32:55	0,96	00:37:14	0,96	00:00:12	0,96
		L <sub>6,3</sub>	02:49:35	0,87	01:24:52	0,87	00:00:43	0,89
Q <sub>7</sub> <sup>a</sup>	Independent	L <sub>7,1</sub>	04:32:13	0,93	01:02:33	0,93	00:00:31	0,92
		L <sub>7,2</sub>	01:32:53	0,93	00:48:54	0,93	00:00:26	0,93
		L <sub>7,3</sub>	07:52:15	0,82	02:36:41	0,82	00:01:16	0,81
Q <sub>8</sub> <sup>a</sup>	Mixed	L <sub>8,1</sub>	04:08:03	0,92	01:34:54	0,92	00:00:32	0,92
		L <sub>8,2</sub>	01:10:33	0,94	00:41:19	0,94	00:00:23	0,93
		L <sub>8,3</sub>	10:04:35	0,78	03:13:04	0,78	00:01:16	0,78
Q <sub>9</sub> <sup>a</sup>	Dependent	L <sub>9,1</sub>	03:55:53	0,92	01:35:44	0,92	00:00:44	0,88
		L <sub>9,2</sub>	01:05:22	0,92	00:35:24	0,92	00:00:27	0,91
		L <sub>9,3</sub>	08:12:41	0,83	02:50:31	0,83	00:01:20	0,82

in successive stages through a decomposition approach [33]. 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 [18]. 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.

**Acknowledgments:** This work is supported by Comisión Nacional de Investigación Científica y Tecnológica–CONICYT – Ministry of Education, Chile, Ph.D. Student Fellowships Folio 21120850.

## References

1. van der Aalst, W.M.P.; van Hee, K.M.; van der Werf, J.M.E.M.; Verdonk, M. Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. *IEEE Computer* **2010**, *43*, 90–93.
2. Gottschalk, F.; van der Aalst, W.M.P.; Jansen-Vullers, M.H.; La Rosa, M. Configurable Workflow Models. *Int. J. Cooperative Inf. Syst.* **2008**, *17*, 177–221.
3. Ghedira, C.; Mezni, H. Through personalized web service composition specification: from BPEL to C-BPEL. *Electronic Notes in Theoretical Computer Science* **2006**, *146*, 117–132.
4. Rosemann, M.; van der Aalst, W.M.P. A configurable reference modelling language. *Inf. Syst.* **2007**, *32*, 1–23.
5. Schunselaar, D.M.M.; Leopold, H.; Verbeek, H.M.W.; van der Aalst, W.M.P.; Reijers, H.A. Configuring Configurable Process Models Made Easier: An Automated Approach. *Business Process Management Workshops*. Springer, 2014, Vol. 202, *Lecture Notes in Business Information Processing*, pp. 105–117.
6. Sharma, D.K.; Rao, V.; others. Configurable Business Process Modeling Notation. *Advance Computing Conference (IACC)*, 2014 IEEE. IEEE, 2014, pp. 1424–1429.

7. van der Aalst, W.M.P. A Decade of Business Process Management Conferences: Personal Reflections on a Developing Discipline. BPM 2012; Barros, A.P.; Gal, A.; Kindler, E., Eds. Springer, 2012, Vol. 7481, LNCS, pp. 1–16.
8. van der Aalst, W.M.P. *Process Mining - Data Science in Action, Second Edition*; Springer, 2016.
9. Buijs, J.C.A.M.; van Dongen, B.F.; van der Aalst, W.M.P. Mining Configurable Process Models from Collections of Event Logs. BPM 2013; Daniel, F.; Wang, J.; Weber, B., Eds. Springer, 2013, Vol. 8094, LNCS, pp. 33–48.
10. Schunselaar, D.M.M.; Verbeek, H.; van der Aalst, W.M.P.; Reijers, H.A. Creating Sound and Reversible Configurable Process Models Using CoSeNets. BIS 2012; Abramowicz, W.; Kriksciuniene, D.; Sakalauskas, V., Eds. Springer, 2012, Vol. 117, LNBIP, pp. 24–35.
11. La Rosa, M.; Dumas, M.; Uba, R.; Dijkman, R.M. Merging Business Process Models. On the Move to Meaningful Internet Systems: OTM 2010; Meersman, R.; Dillon, T.S.; Herrero, P., Eds. Springer, 2010, Vol. 6426, LNCS, pp. 96–113.
12. Mendling, J.; Simon, C. Business Process Design by View Integration. BPM 2006; Eder, J.; Dustdar, S., Eds. Springer, 2006, Vol. 4103, LNCS, pp. 55–64.
13. Gottschalk, F. Configurable process models. PhD thesis, Eindhoven University of Technology, 2009.
14. La Rosa, M.; Dumas, M.; Uba, R.; Dijkman, R.M. Business Process Model Merging: An Approach to Business Process Consolidation. *ACM Trans. Softw. Eng. Methodol.* **2013**, *22*, 11.
15. Gottschalk, F.; van der Aalst, W.M.P.; Jansen-Vullers, M.H. Merging Event-Driven Process Chains. OTM Conferences (1). Springer, 2008, Vol. 5331, *Lecture Notes in Computer Science*, pp. 418–426.
16. La Rosa, M.; van der Aalst, W.M.P.; Dumas, M.; ter Hofstede, A.H.M. Questionnaire-based variability modeling for system configuration. *Software and System Modeling* **2009**, *8*, 251–274.
17. Buijs, J.C.A.M. Environmental permit application process ('WABO'), CoSeLoG project. <https://data.4tu.nl/repository/uuid:26aba40d-8b2d-435b-b5af-6d4bfbfd7a270>, 2014.
18. Buijs, J.C.A.M. Flexible Evolutionary Algorithms for Mining Structured Process Models. PhD thesis, Eindhoven University of Technology, 2014.
19. van der Aalst, W.M.P.; Buijs, J.C.A.M.; van Dongen, B.F. Towards Improving the Representational Bias of Process Mining. SIMPDA 2011; Aberer, K.; Damiani, E.; Dillon, T.S., Eds. Springer, 2011, Vol. 116, LNBIP, pp. 39–54.
20. Buijs, J.C.A.M.; van Dongen, B.F.; van der Aalst, W.M.P. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. OTM 2012; Meersman, R.; Panetto, H.; Dillon, T.S.; Rinderle-Ma, S.; Dadam, P.; Zhou, X.; Pearson, S.; Ferscha, A.; Bergamaschi, S.; Cruz, I.F., Eds. Springer, 2012, Vol. 7565, LNCS, pp. 305–322.
21. Leemans, S.J.J.; Fahland, D.; van der Aalst, W.M.P. Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. PETRI NETS 2013; Colom, J.M.; Desel, J., Eds. Springer, 2013, Vol. 7927, LNCS, pp. 311–329.
22. Leemans, S.J.J.; Fahland, D.; van der Aalst, W.M.P. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. BPM 2013; Lohmann, N.; Song, M.; Wohed, P., Eds. Springer, 2013, Vol. 171, LNBIP, pp. 66–78.
23. La Rosa, M.; Dumas, M.; ter Hofstede, A.H.M.; Mendling, J. Configurable multi-perspective business process models. *Inf. Syst.* **2011**, *36*, 313–340.
24. Munoz-Gama, J. *Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes*; Springer, 2016.
25. Mitchell, M. *An Introduction to Genetic Algorithms*; MIT Press: Cambridge, MA, USA, 1998.
26. Deb, K.; Deb, D. Analysing mutation schemes for real-parameter genetic algorithms. *IJAISC* **2014**, *4*, 1–28.
27. Bäck, T.; de Graaf, J.M.; Kok, J.N.; Kusters, W.A. Theory of Genetic Algorithms. In *Current Trends in Theoretical Computer Science*; 2001; pp. 546–578.
28. Michalewicz, Z. *Genetic algorithms and data structures - evolution programs (3. ed.)*; Springer, 1996.
29. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms (3. ed.)*; MIT Press, 2009.
30. Gottschalk, F.; Wagemakers, T.A.C.; Jansen-Vullers, M.H.; van der Aalst, W.M.P.; La Rosa, M. Configurable Process Models: Experiences from a Municipality Case Study. *Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009, Proceedings, 2009*, pp. 486–500.



31. Adriansyah, A. Aligning observed and modeled behavior. PhD thesis, Eindhoven University of Technology, 2014.
32. Van Dongen, B. BPI Challenge 2015. <http://dx.doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1>, 2015.
33. de Leoni, M.; Munoz-Gama, J.; Carmona, J.; van der Aalst, W.M.P. Decomposing Alignment-Based Conformance Checking of Data-Aware Process Models. OTM Conferences. Springer, 2014, Vol. 8841, *Lecture Notes in Computer Science*, pp. 3–20.