ProDiGen: mining complete, precise and minimal structure process models with a genetic algorithm

Borja Vázquez-Barreiros[a,*], Manuel Mucientes[a], Manuel Lama[a]

[a] *Centro de Investigación en Tecnoloxías da Información (CiTIUS)*
University of Santiago de Compostela Spain

Abstract

Process discovery techniques automatically extract the real workflow of a process by analyzing the events that are collected and stored in log files. Although in the last years several process discovery algorithms have been presented, none of them guarantees to find complete, precise and simple models for all the given logs. In this paper we address the problem of process discovery through a genetic algorithm with a new fitness function that takes into account both completeness, precision and simplicity. ProDiGen (Process Discovery through a Genetic algorithm) includes new definitions for precision and simplicity, and specific crossover and mutation operators. The proposal has been validated with 39 process models and several noise levels, giving a total of 111 different logs. We have compared our approach with the state of the art algorithms; non-parametric statistical tests show that our algorithm outperforms the other approaches, and that the difference is statistically significant.

Keywords: Genetic mining, process discovery, Petri nets, process mining.

1. Introduction

In the last decade, a great effort for developing technologies to automate the execution of processes has been made in different application domains such as industry, education or medicine [10]. In this context, a process is understood as a collection of tasks —or activities— with coordination requirements among them [32]. These tasks are performed by a set of actors to achieve the purpose of the process. For instance, in education the learning design of a course is a process where learners must undertake a sequence of learning activities, e.g., posting in forums, making exercises and exams, etc., in order to achieve the pedagogical objectives of the course. Typically, these processes have a detailed description, i.e., there is a design of the process where its activities and the actors participating in these steps are clearly described. However, even in this situation there might be differences between what is actually happening and what is predefined in the process. For instance, following with the example of the education domain, learners can undertake additional learning activities —like check the bibliography or interact with other learners— apart from those that were explicitly specified in the learning process designed by a teacher.

*Corresponding author
Email addresses: borja.vazquez@usc.es (Borja Vázquez-Barreiros), manuel.mucientes@usc.es (Manuel Mucientes), manuel.lama@usc.es (Manuel Lama)

At this point, Process Mining (PM) techniques are needed to get information about *what is really happening* in the execution of a process, and *not what the people think it is happening* [34]. Typically these techniques use the log files that collect information about the events detected and stored by the information system in which the process has been performed. PM techniques can be classified in three different groups [29]. The first one is *process discovery*, which aims to retrieve the process model that represents the behavior recorded in an event log. These algorithms are used to discover the underlying process that has been followed by users to achieve an objective. The second class of process mining techniques is *conformance checking*, where a process model is compared with a log of the same process to analyze and quantify the deviations between the observed and the real behavior, as recorded in the log. These techniques are focused on providing an *understanding* of the real processes that take place in an organization. The third group is *enhancement*, where a process model is dynamically modified or extended based on the information from the log. In this paper, we address the problem of process discovery. More specifically, our interest lies in the control-flow of the recorded events, i.e., the ordering of the activities.

Over the last decade, several papers have dealt with the discovery problem in process mining [34, 9, 41, 42, 35, 4, 40, 8]. Unfortunately, existing techniques may produce models that are unable to replay the log, may produce complex and unreadable models, or may retrieve erroneous models. For instance, those approaches based just on the local information provided by the log [34] are only capable to overcome specific weak points in the field under some conditions —completeness and noise-free logs—, like short loops [9], non-free-choice constructs [41] or invisible tasks [42], but not all at once. Other papers solve the process discovery problem with search-based approaches, based on heuristics or on theory of regions [35, 4, 40, 8]. Some techniques guarantee sound models [4], others guarantee the rediscoverability of the main behavior of the log [40], some guarantee completeness [35] and others can tackle all the different and main pattern constructs at once [8] but leaving simplicity aside. Nevertheless, there is no discovery algorithm that can tackle all the different structures at once, and that can find complete, precise and simple models. Furthermore, many of them have problems while dealing with noise.

In this paper we present ProDiGen[1] (Process Discovery through a Genetic algorithm), a process discovery algorithm that searches complete, precise and simple models. The contributions of this proposal are:

1. A hierarchical fitness function that takes into account completeness, precision and simplicity.
2. A new definition of precision based on the log and the mined model.
3. A new definition of simplicity based on the mined model.
4. A crossover operator that selects the crossover point from a Probability Density Function (PDF) generated from the errors of the mined model.
5. A mutation operator guided by the causal dependencies of the log.

The proposal has been tested with 39 models with several noise levels and different degrees of complexity, giving a total of 111 different logs. Moreover, we have compared our approach with four of the state of the art algorithms using

[1] `http://tec.citius.usc.es/SoftLearn/ProDiGen_ins.html`

a collection of conformance checking metrics. The results of the comparison have been validated with non-parametric statistical tests.

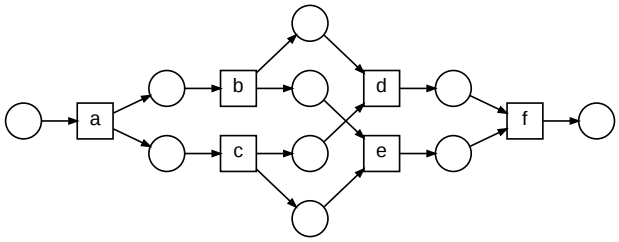
The remainder of this paper is structured as follows. Sec. 2 introduces the process discovery problem, and Sec. 3 describes the different approaches that have already been proposed. Then, Sec. 4 presents the proposed genetic algorithm for process discovery. Sec. 5 shows the obtained results and the comparison with other approaches, and, finally, Sec. 6 points out the conclusions.

2. Process Discovery

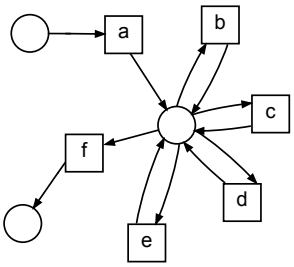
The goal of process discovery is to obtain a process model that specifies the relations between tasks —or activities— in an event log. The basic assumption is that there is a process model that generates the log with the following rules: (i) each event is a well-defined step in some process; (ii) each event is related to a particular case; and (iii) the events are sequentially saved no matter the type of pattern behind. Fig. 1a represents a simple log with 18 events, 6 different activities and performed by three different users.

User	Event	Lifecycle	Timestamp
Pablo	Introductory class	complete	07-03-2013:10:00
Borja	Introductory class	complete	09-03-2013:16:00
Pablo	Finite Automaton	complete	11-03-2013:19:30
Pablo	Regular Grammar	complete	17-03-2013:15:28
Borja	Finite Automaton	complete	20-03-2013:10:12
Manuel	Introductory class	complete	20-03-2013:11:42
Borja	Regular Grammar	complete	21-03-2013:14:34
Manuel	Regular Grammar	complete	23-03-2013:09:21
Pablo	Context-Free Grammar	complete	01-04-2013:12:36
Manuel	Finite Automaton	complete	01-04-2013:15:54
Borja	Pushdown Automaton	complete	04-04-2013:17:20
Manuel	Context-Free Grammar	complete	06-04-2013:20:00
Pablo	Pushdown Automaton	complete	21-04-2013:11:02
Borja	Context-Free Grammar	complete	22-04-2013:15:09
Manuel	Pushdown Automaton	complete	28-04-2013:17:45
Pablo	Exam	complete	06-05-2013:18:45
Manuel	Exam	complete	06-05-2013:19:01
Borja	Exam	complete	06-05-2013:19:22

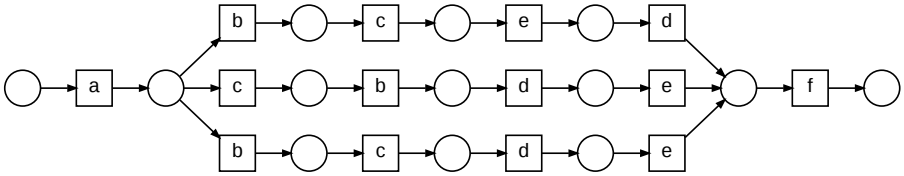
(a) *An event log.*



(b) *A complete, specific and general model: Completeness +, Precision +, Generalization +, Simplicity +.*



(c) *A complete over-general model: Completeness +, Precision -, Generalization +, Simplicity +.*



(d) *A complete over-specific model: Completeness +, Precision +, Generalization -, Simplicity -.*

Figure 1: Discovery of a process model prioritizing different objectives. The models are represented as Petri nets. The name of the activities are: *Introductory class* (a), *Finite Automaton* (b), *Regular Grammar* (c), *Context-free grammar* (d), *Pushdown automaton* (e) and *Exam* (f)).

To discover the underlying process, the proposal presented in this paper only needs the list of events in the log and their corresponding case identifiers. Other process discovery techniques can use more information, like the timestamp [33, 5] or the data attributes that affect the routing of the cases [25]. The event log of Fig. 1a shows the process

instances of three different students during the course *Automata Theory and Formal Languages*. In this log, all the students started with an introductory class, and then they attended to four different lessons (*finite automaton*, *regular grammar*, *context-free grammar* and *pushdown automaton*). Once all these four lessons are finished, they make an exam. Using the information provided by the log, the discovery of a process model can take place with different objectives [24]:

- **Completeness**. Measures how much of the behavior observed in the log can be reproduced by the mined model. A model is complete when it can reproduced all the behavior in the log.
- **Precision**. The objective of this metric is to avoid overly general models. The model in Fig. 1c is able to parse all the traces in the log. However, it is an overly general model, as it allows more behavior than the contained in the log. For example, one valid trace for this model is the possibility to make the exam after attending only to the introductory class.
- **Generalization**. This metric tries to prevent overly precise models. For instance, although the model in Fig. 1d is complete, it is overly specific because it merely recreates each one of the traces of the log, not allowing extra behavior. When mining models, there is a tradeoff between precision an generalization, which can be compared to the bias-variance tradeoff.
- **Simplicity**. Indicates to models with a minimal structure that reflects the behavior in the log. For example, the model in Fig. 1d creates one path for every possible trace of the log, i.e., it is not simple as it contains several duplicated tasks, which make the model difficult to read.

Fig. 1b shows a model that takes into account the four objectives. This model is well-structured, providing not only the behavior shown in the log, but allowing also the trace *"Intro, Regular Grammar, Finite Automaton, Pushdown Automaton, Context-Free Grammar, Exam"*. Moreover, the model does not allow random behavior.

3. Related Work

Since Cook and Wolf [7] coined the term process discovery, and later on, Agrawal et al. [1] applied this idea in the context of workflow management systems, dozens of process discovery methods have been proposed [29]. Although some mining techniques use a specific target model for control-flow discovery [16], most of the process discovery algorithms are based on Petri nets [23]. These algorithms can be classified in four groups [37]:

- **Abstraction-based algorithms**. This type of algorithms is based on a complete and noise free-log. All of them are derived from the α-algorithm [34] to address some of its drawbacks. The limitations solved with these extensions are: short loops (α^+-algorithm [9]), non-free-choice constructs (α^{++}-algorithm [41]), invisible tasks ($\alpha^\#$-algorithm [42]) and duplicate tasks (α^*-algorithm [20]). Despite the different extensions, none of the

abstraction-based algorithms can tackle all the complex constructs at once. In general, this type of algorithms focuses on simplicity, retrieving very simple models but with poor completeness.

- **Heuristics-based algorithms**. In [40, 39], Weijters et al. presented the Heuristics Miner, an extension of the α-algorithm but taking into account the frequency of ordering relations. One of its main advantages is its ability to handle noise based on a set of thresholds. Thus, this method is appropriate for identifying the main behavior registered in the log, excluding duplicate tasks and some non-free-choice constructs. DWS [14] is an extension of Heuristics Miner that identifies different variants of a process model by clustering similar log traces. Another extension of the Heuristics Miner was presented in [5]. It takes into account the timestamp of the activities, expressing the activity as time intervals instead of single events. Heuristics-based algorithms use replay fitness (completeness) as their guiding principle, but do not guarantee optimal results in terms of completeness, as they only focus on the main behavior.

- **Search-based algorithms**. So far, the previously described algorithms are based on local information and therefore, they cannot discover some constructs like non-free-choices. To overcome this situation, the search-based algorithms perform a global search based on an abstraction from local properties like ordering relations. With Genetic Miner, Alves de Medeiros et al. [8] proved that it is possible to mine all common constructs and be robust to noise, all at once, but it cannot ensure simple models as some of the mined solutions have implicit places or needless arcs. Another approach recently proposed [4] guides its search taking into account a balance between the four objectives described in Sec. 2, considering only block-sructured solutions.

- **Algorithms based on theory of regions**. They can be classified in two groups based on their behavioral process specification: *state-spaced* and *language* based. The *state-based* algorithms perform two steps: first, they build a transition system [31] —a set of states and transitions between states—, and then they construct a Petri net according to that transition system [2, 6, 28, 11]. This group of algorithms focuses on the synthesis of a Petri net whose reachability graph is similar to the transition system. As discussed in [35] and [31] the main problem of this solution is the non-trivial construction of the state information from a log, because usually logs almost never carry state information. In contrast, *language-based* algorithms assume that the log contains words (traces) of a specific language —the activities are the letters—, whereas the target net allows just words of this language. For *language-based* algorithms, in [3], the authors distinguish between two methods to derive Petri nets from event logs, (i) using a *basis representation*, which cannot tackle duplicate tasks or non-free-choice constructs; and (ii) using *separating representation* [21, 3] to mine duplicate tasks but not non-free-choice constructs. Both approaches lead to a model overfitting the log due to the restrictive assumptions about process logs. Additionally, the number of places introduced by both approaches is theoretically high. To overcome these drawbacks, in [35], authors propose to use Integer Linear Programming to avoid overffited models and minimize the upper bound number of places. However, because no assumptions are made about the completeness of the log, the solution might be an underfitted model, allowing for much more extra behavior. These

algorithms usually guarantee a perfect replay fitness, but, unfortunately, these techniques still have problems with incomplete behavior.

A method that does not fit in any of these categories and is based on **inductive logic programming** was presented by Goedertier et al. [13]. They designed an algorithm that introduces artificially generated negative events, i.e., traces that describe a particular path that is not allowed for a process. Unfortunately, event logs hardly ever contains information about disallowed behavior.

In summary, a large amount of work has been done in this specific area by addressing solutions from different points of view. Unfortunately, none of the techniques can retrieve models with high levels of completeness, but being as precise and simple as possible. Furthermore, none of these techniques can handle all the different control constructs and noise all at once, but ensuring completeness, precision and simplicity. Hence, we propose a more elaborate approach based on the idea of Genetic Miner [8] to overcome these drawbacks.

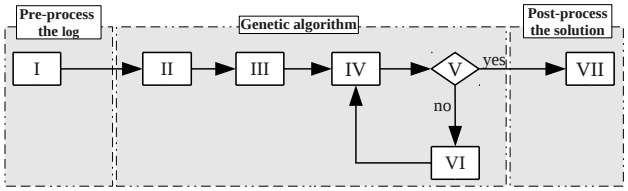
4. ProDiGen: Process Discovery through a Genetic algorithm

Our proposal (ProDiGen) is inspired in the Genetic Miner algorithm [8], which can tackle all the different constructs at once. However, Genetic Miner has several drawbacks: (i) it is not able to mine complete and precise models when they have many interleaving situations; (ii) the mined models are usually hard to interpret and unnecessarily complex; and (iii) it needs many generations to converge to a solution.

Fitness	The fitness is hierarchical and takes into account the completeness, precision and simplicity of the mined model.
Precision	Definition of a new method to measure the precision of a model.
Simplicity	Definition of a new method to measure the simplicity of a model.
Initialization	ProDiGen incorporates the result of the Heuristics Miner [40] into the initial population.
Selection	Binary tournament selection.
Replacement	It selecs the best individuals of a joint population of parents and offspring. The reinitialization criterium is based on the improvement of the population.
Crossover	The crossover operator is guided by a Probability Density Function (PDF) generated from the errors of the mined model.
Mutation	The mutation operator is guided by the causal dependencies of the log.

Table 1: Differences between ProDiGen and Genetic Miner.

The drawbacks of Genetic Miner are caused by: (i) a weighted fitness function that combines completeness and precision in an inadequate way; (ii) the precisions of the models are not very informative as they depend on the precisions of the other individuals of the population; (iii) the fitness function does not take into account the simplicity of the model; and (iv) the genetic operators are executed in a completely random way, without taking advantage of the information of the log and the errors of the mined model during the parsing of the traces.



Step	Description
I	Pre-process the log
II	Calculate the dependencies between tasks
III	Build the initial population
IV	Evaluate each solution
V	Stopping conditions fulfilled?
VI	Generate the new population
VII	Post-process the best individual

Figure 2: Main steps of ProDiGen.

The differences between ProDiGen and Genetic Miner are summarized in Tab. 1. As can be seen, almost all the mains steps of the genetic algorithm have been changed in order to overcome the previously discussed drawbacks. In particular, one of the major changes takes place in the hierarchical fitness function, where completeness, precision and simplicity are considered for the evaluation of an individual. Additionally, we introduce heuristics to guide the genetic operators, focusing the search on those parts of the mined model that have errors and, also, looking for new models that are supported by the information in the log.

The main steps of ProDiGen are shown in Fig. 2. The algorithm has three phases: (i) a *pre-processing* of the log, which groups and filters out the noise in the log; (ii) the core of ProDiGen is the *genetic algorithm* phase; and (iii) a *post-processing* of the mined model, to prune unused and infrequent arcs. The genetic algorithm is described in 4.1 and both the pre-processing and post-processing steps are described in Sec. 4.2.

Algorithm 1: Genetic algorithm for process discovery.

```
1  Initialize population
2  Evaluate population
3  t = 1, timesRun = initialTimesRun, restarts = 0
4  while t ≤ maxGenerations && restarts < maxRestarts do
5      Selection
6      Crossover
7      Mutation
8      Evaluate new individuals
9      Replace population
10     t = t + 1
11     if bestInd(t) == bestInd(t − 1) then
12         timesRun = timesRun − 1
13     if none of the individuals of the population have been replaced then
14         timesRun = timesRun − 1
15     if timesRun < 0 then
16         Reinitialize population
17         Evaluate population
18         timesRun = initialTimesRun, restarts = restarts + 1
```

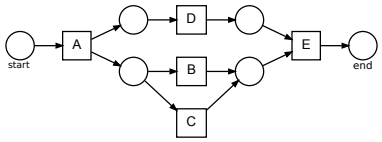
4.1. Genetic algorithm

Algorithm 1 describes the genetic algorithm. The first three steps correspond to an initialization, where *t* represents the number of iterations, *timesRun* is used to detect situations in which the search gets stuck, and *restarts* counts the number of executed reinitializations. The evolution cycle of the algorithm starts at Alg. 1:4. This part will be repeated until the stopping criterion is fulfilled. The main steps of the iterative part are the selection of the individuals, the crossover and mutation operations to generate new individuals, their evaluation, the replacement of the population, and the analysis of the population to detect blockages in the search process. All of these steps are described in detail in the next sections.

4.1.1. Internal representation

Each individual of the population codifies a workflow using the causal matrix representation [8], which can map any Petri net in terms of causal dependencies. The causal matrix has a row for each task *t* in the log, and two columns corresponding to the inputs —I(t)— and outputs —O(t)— of each task *t* (see Fig. 3 for an example). Those tasks in

the same subset of I(t) have an OR-join relation, and those on different subsets an AND-join relation. On the other hand, tasks in the same subset of O(t) have an OR-split relation and those in different subsets an AND-split relation.



(a) *Example of a Petri net.*

Task	I(Task)	O(Task)
A	{}	{{D},{C B}}
B	{{A}}	{{E}}
C	{{A}}	{{E}}
E	{{D},{B,C}}	{}
D	{{A}}	{{E}}

(b) *Causal matrix of the Petri net.*

Figure 3: Mapping of a petri net into a causal matrix

4.1.2. Initialization

The initialization follows the heuristic approach described in [8], which is based on the causality relations between tasks. Moreover, we also add to the initial population an individual mined with the Heuristics Miner approach [40]. It is important to notice that the inclusion of the Heuristics Miner individual does not modify the best mined model of ProDiGen in any of the 111 logs tested in the results section. Nevertheless, the inclusion of this individual in the initial population *speeds up* the iteration at which the best individual is found, as the main dependency relations are captured by Heuristics Miner —these dependencies are more robust than the ones defined in [8]— and then, with ProDiGen, the different inputs and outputs bindings are optimized.

4.1.3. Evaluation

Individuals of the population are evaluated with a hierarchical fitness function that takes into account completeness, precision and simplicity.

Completeness. A natural definition for completeness would be the number of properly parsed traces divided by the total number of event traces . However, this definition is not able to distinguish between two individuals that cannot process a trace; e.g., one of them because an arc is missing and the other one because the model is totally incorrect. For this reason, we use the definition of completeness (C_f) described in [8], which takes into account the number of correctly parsed tasks[2], but also the number of missing and not consumed tokens of the Petri net encoded in the individual —each missing or not consumed token represents a failure.

Precision. The measurement of the precision of a mined model is difficult, as precision has to detect the extra behavior, i.e., paths in the model that are not represented in the log. Therefore, our definition of precision considers all the activities that are enabled while an individual parses the log:

$$P_f(L,\ CM) = \frac{1}{allEnabledActivities\,(L,\ CM)} \tag{1}$$

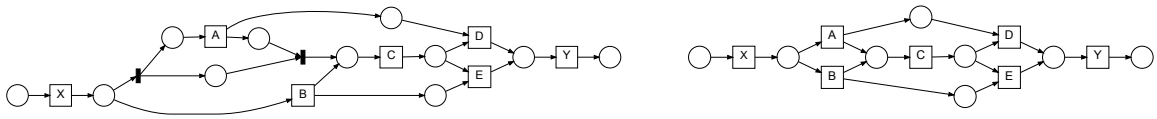
[2]If a task from an individual does not have the proper input arcs, that task will be incorrectly parsed when reproducing the log, as its input conditions are not fulfilled.

where *allEnabledActivities* is the number of enabled activities when a log L is processed by an individual CM. *allEnabledActivities* is evaluated by counting the number of enabled activities after firing each activity of a trace. This process is performed for every trace in the log. Thereby, with this definition for the precision ProDiGen punishes those models with too many enabled activities, as each enable activity represents a possible path for extra behavior. Contrary to [8], we do not consider the rest of the population in order to compute the precision of each individual, which can evolve without taking into account the precision of the rest of the population.

Simplicity. The third dimension of the fitness is simplicity, which measures the complexity of a mined model based on the number of causal relations of an individual:

$$S_f(CM) = \frac{1}{\sum_{t \in CM} \left(\sum_{\Phi \in I(t)} |\Phi| + \sum_{\Psi \in O(t)} |\Psi| \right)} \tag{2}$$

where t is a task of the causal matrix CM, Φ is an element of I(t) —the tasks in Φ have an OR-join relation—, and Ψ is an element of O(t) —the tasks in Ψ have an OR-split relation. Therefore, the simplicity counts the number of causal relations of the model using the cardinality of the input and output subsets of the causal matrix.



(a) *Mined model with needless branches.* (b) *Simplest mined model.*

Figure 4: Two possible solutions with the same completeness and precision

We illustrate the relevance of simplicity to mine the original model with a simple example of two traces repeated three times: $<< X, B, C, E, Y >^3$, $< X, A, C, D, Y >^3>$. Fig. 4 shows two mined models that discover the non-free-choice[3] construction, and have the same completeness and precision: (i) both can parse exactly the same tasks, i.e., *completeness* = 1.0; and (ii) they enable exactly the same number of tasks during the parsing (36), thus *precision* = 1/36. However, the model in Fig. 4a has a *simplicity* = 1/24 while the model in Fig. 4b has a *simplicity* = 1/20 and, therefore, the second one is a better model. The difference between these two solutions —in terms of simplicity— is caused by the output function of the task X and the input function of the task C: (i) the causal matrix of the model in Fig. 4a has $O(X) = \{\{A, B\}, \{B, C\}\}$, and $I(C) = \{\{A, B\}, \{B, X\}\}$ which increases the complexity of the model by 8; (ii) the causal matrix of the model in Fig. 4b has $O(X) = \{\{A, B\}\}$, and $I(C) = \{\{A, B\}\}$ which increases the complexity of the model by 4.

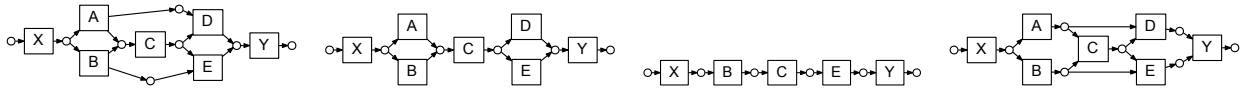
Fitness. ProDiGen uses completeness, precision and simplicity to evaluate the mined models. However, instead of combining these three objectives in a weighted sum —which requires the definition of a new weight parameter for

[3] A non-free-choice construction is a mixture of a synchronization and a choice [8]. For example, in Fig. 4, the execution of D or E depends on whether the task A or B has been executed.

each criteria— it defines a hierarchical fitness function that establishes priorities among the objectives:

$$F(a) > F(b) \iff \{C_f(a) > C_f(b)\} \vee \{C_f(a) = C_f(b) \wedge P_f(a) > P_f(b)\} \vee \{C_f(a) = C_f(b) \wedge P_f(a) = P_f(b) \wedge S_f(a) > S_f(b)\} \quad (3)$$

where $F(a)$, $C_f(a)$, $P_f(a)$ and $S_f(a)$, are respectively the *fitness*, *completeness*, *precision* and *simplicity* of a process model a. The advantage of using this hierarchical fitness function over a weighted fitness function is that, during the first stage of the evolutionary process, the GA focuses the search on those individuals that are complete. Once these individuals become representative in the population, the second level of the hierarchy takes the control, modifying the models that are complete in order to improve their precision. Finally, in the third stage, the fitness function guides the GA to improve the simplicity of those models that are both complete and precise.



(a) $C_f = 1.0$, $P_f = 1/12$, $S_f = 1/20$. (b) $C_f = 1.0$, $P_f = 1/14$, $S_f = 1/16$. (c) $C_f = 0.2$, $P_f = 1/18$, $S_f = 1/8$. (d) $C_f = -0.2$, $P_f = 1/6$, $S_f = 1/20$.

Figure 5: Four different models prioritizing the different search criteria.

If we change the hierarchical order of the fitness measure, the algorithm may find a different solution, as completeness, precision and simplicity are three opposed objectives. Fig. 5 shows four different models that can be found mining two traces: $<< X, B, C, E, Y >, < X, A, C, D, Y >>$ but prioritizing different objectives. For example, both the models of Fig. 5b and Fig. 5a have the same completeness but, in order to achieve a better simplicity, the solution of Fig. 5a retrieves a lower precision than the solution of Fig. 5b. If we want to retrieve a simple model (Fig. 5c) the solution will have a lower completeness. On the other hand, if we want to retrieve a model with a better precision (Fig. 5d), both the completeness and the simplicity will be lower.

4.1.4. Genetic operators

The process to create new individuals starts with the selection phase. ProDiGen uses as selection mechanism a binary tournament selection, in which two individuals of the population are randomly picked —with replacement—, and the best of them is selected. Then, each pair of individuals in the selected population is crossed and mutated.

Crossover. The crossover operator replaces causality relations of an individual with causality relations of another individual. As the process models are represented through causal matrices, and the size of the causal matrix increases with the number of activities in the log, the number of possible crossover points could be really large. We have noticed that picking the crossover point at random produces a poor performance of the crossover operator, as most of *the offspring have a fitness lower than their parents*. ProDiGen makes the selection of the activity that is going to be crossed using a non uniform PDF. This PDF assigns a null probability of being selected to those activities that have been correctly fired during the parsing of the traces in the log. On the other hand, those activities that were incorrectly

Algorithm 2: Pseudo-code for the crossover operator.

```
1  r ← getRandomNumber() // returns a random number between [0,1)
2  if r < crossoverRate then
3      incorrectlyFiredActivities ← ∅
4      if fitness(parent_1) >= fitness(parent_2) then
5          incorrectlyFiredActivities ← set of incorrectly fired activities of parent_1
6      else
7          incorrectlyFiredActivities ← set of incorrectly fired activities of parent_2
8      if incorrectlyFiredActivities ≠ ∅ then
9          crossoverPoint ← randomly select an activity t from incorrectlyFiredActivities
10     else
11         crossoverPoint ← randomly select an activity t from the bag of all possible tasks in the log
12     offspring_1, offspring_2 ← doCrossover(parent_1, parent_2, crossoverPoint)
13     Repair offspring_1 and offspring_2
```

fired receive a uniform probability —inversely proportional to the number of incorrectly parsed activities— of being crossed.

Alg. 2 summarizes the behavior of the crossover operator. By incorrectly fired activities we mean (i) activities that need extra tokens in their inputs to be fired, i.e, tasks that do not have the correct input arcs, and (ii) activities that have left tokens in their outputs after the parsing, i.e., activities that do not have the correct output arcs. This process generates, for each individual, a bag of *incorrectlyFiredActivities*. Thereby, the crossover point is selected from the set of *incorrectlyFiredActivities* of the fittest parent (Alg. 2:4). Note that if the fittest individual has a completeness equal to 1, the set of *incorrectlyFiredActivities* of the fittest individual is empty (Alg. 2:8); thereby the crossover point is randomly chosen from the bag of all the possible tasks in the log (Alg. 2:11).

After the crossover point is selected, the crossover (Alg. 2:12) is performed as defined in [8]. The following example illustrates how the crossover operator works. Let's suppose that the crossover point is task A, and the input of the individual 1 for that task is $I_1(A) = \{\{D\}, \{B\}\}$ and for individual 2 is $I_2(A) = \{\{D, B\}, \{C\}\}$. First, both input sets are split randomly. This process generates four sets, two for each input function:

- $I_1^1(A) = \{\{D\}\}$ and $I_1^2(A) = \{\{B\}\}$.
- $I_2^1(A) = \{\{D, B\}\}$ and $I_2^2(A) = \{\{C\}\}$.

Then, these four sets are swapped — $I_1^1(A)$ with $I_2^2(A)$ and $I_2^1(A)$ with $I_1^2(A)$— in order to generate the new input sets for the task A of the offspring. During this process, each subset of these sets can be added as a new subset into the other set or merged with another subset of the other set. For instance, the subset of $I_1^1(A)$ can be added as a new subset of $I_2^2(A)$, resulting in $I_1'(A) = \{\{D\}, \{C\}\}$; and the subset of $I_2^1(A)$ can be joined with the existing subset of $I_1^2(A)$, resulting in $I_2'(A) = \{\{D, B\}\}$ —subsets do not allow duplicate tasks. Finally, this process is repeated for the output set of the crossover task A.

Note that when adding/removing causal relations from an input/output (I/O) set of a task t —being t the *crossoverPoint*— there may be inconsistencies. For instance, a task t' does not appear in the output set of the task t, but the input set of t' contains the task t. Therefore, after an I/O set is modified, we have to check the consistency of the individual (Alg.

2:13). This process first checks those relations that were removed during the modification of the individual, and after that, it checks those relations that were created. This prevents to create again a relation that was previously removed.

Mutation. The mutation operator modifies the causality dependencies of the individual by adding or removing relations. The mutation operator may perform one of the following three actions to the input/output set of a task: (i) randomly add a task *t*' to input or output sets of a task *t*; (ii) randomly remove a task *t*' from the I/O sets of a task *t*; and (iii) randomly redistribute the elements from the I/O sets of a task *t*.

Algorithm 3: Pseudo-code for the mutation operator.

```
1  while the individual does not change do
2      Randomly choose one task t in the individual
3      mutationType ← getRandomNumber() // returns a random number between [0,1)
4      if mutationType < 1/3 then
5          Randomly add a new task t' to I(t), being t' a task from inputDependencies(t)
6          if getRandomNumber() < 1/2 then
7              Randomly choose one subset X ∈ I(t) and add the task t' to X
8          else
9              Create a new subset X, add the task t' to X, and add X to I(t)
10     else if mutationType < 2/3 then
11         Randomly choose one subset X ∈ I(t) and remove a task t' from X, where t' ∈ X. If X is empty after this operation, exclude X from I(t)
12     else
13         Randomly redistribute the elements from I(t)
14     Repeat from line 3, but using O(t) instead of I(t) and outputDependencies(t) instead of inputDependencies(t)
15     Repair the individual
```

There are four differences between our mutation operator and the one used in Genetic Miner [8]: (i) the individual is iteratively mutated until it is different from its parent —a mutation could generate an individual equal to its parent due to a reparation; (ii) only one task is affected by the mutation operator; (iii) individuals are always forced to mutate —the mutation probability is 1; and (iv) the task t' added to the I/O set of a task t must belong to the set of tasks that have an input/output dependency with t. The major goal of these modifications is to avoid duplicate individuals within the same population, or at least minimize its duplicates. Hence, although the offspring are equal to their parents after the crossover, we force each offspring to mutate until it changes, creating different individuals with new features. With these modifications, we have a more diverse population.

Alg. 3 describes in detail the mutation operator. It uses two sets for the addition of a new task: *outputDependencies(t)* and *inputDependencies(t)*. Both sets are created when calculating the dependencies between tasks at the first stages of the algorithm. ProDiGen uses these sets to reduce the set of tasks that are appropriate to be inserted in an I/O set, preventing the inclusion of a new task that never appears in a trace of t within the log. A first approach could be to include in the dependencies sets those tasks that have a dependency with t as calculated in the initialization phase. However, if we only take into account these dependencies, there will be not enough new material to discover, for instance, the non-free-choice constructs. Therefore, *inputDependencies(t)* will be the set of tasks appearing before t in any trace of the log and, in the same way, *outputDependencies(t)* will be the set of activities that appear after t in any trace of the log. In this way, the mutation operator focuses only on those regions of the search space that represent information contained in the log. As a result, the success of the mutation operator increases, finding better offspring.

Again, after each mutation the individual has to be repaired (Alg. 3:15) following the same strategy as explained in Sec. 4.1.4.

4.1.5. Replacement

At each iteration, the algorithm generates N offspring, being N the size of the population. These offspring and the parent population —current population— are joined and sorted —using the fitness— generating a $2N$-size population, and then the replacement operator selects the N best individuals. In order to maintain a diverse population, those repeated individuals are placed at the bottom of the ranking, keeping one representative in the original ranking position.

4.1.6. Reinitialization

A reinitialization takes place when the value of *timesRun* goes under 0 (Alg. 1:15), which indicates that the search process was not improving in the last iterations. This situation is detected in two ways. The first one (Alg. 1:11) is when the new population of an iteration has no new individuals —in comparison with the initial population of that iteration. The second indicator (Alg. 1:13) is the fact that the best individual does not improve. Each time that one of these situations is detected, *timesRun* decreases. The initial population after a reinitialization is generated in the same way as in the initialization stage. Moreover, ProDiGen also includes in the new population a mutation of the best individual of the last iteration. The maximum number of reinitializations is limited, and when it reaches the threshold (*maxRestarts*) ProDiGen ends.

4.2. Pre-processing and post-processing steps

Noise can be defined as a low-frequent incorrect behavior in the log [37]. The main difficulty to deal with noise is that the logs —usually— contain only positive examples, i.e., there is no explicit information about the characteristics of the noisy traces. Additionally, there is low-frequent correct behavior in the log that cannot be easily distinguished from the low-frequent incorrect cases.

ProDiGen explicitly handles the noise in two phases: (i) a pre-processing of the log; and (ii) a post-processing of the mined model. In the pre-processing of the log, ProDiGen groups all the traces that are equal, and calculates the normal distribution of the frequency of the traces $\mathcal{N}(\mu, \sigma)$, where μ and σ are respectively the weighted mean and the weighted standard deviation. Finally, all those traces with $frequency(trace) < \mu - \xi\sigma$ will be removed, where ξ is a parameter. Therefore, ProDiGen eliminates those traces that are infrequent, and the threshold depends on the characteristics of the log.

The post-processing stage of ProDiGen consists in a post-pruning over the mined model. It removes those arcs that are used less frequently than a certain threshold, being this threshold a percentage of the frequency of the most used arc. This post-pruning was also applied in other process discovery techniques [8, 17, 15].

5. Experimentation

ProDiGen has been validated with 111 different logs. We have classified these tests in two different groups: (i) 18 process models to generate logs with five noise levels —0%, 1%, 5%, 10%, and 20%—, which results in 90 logs; and (ii) 21 unbalanced logs, i.e., logs that contain traces with very different frequencies, which correspond with 21 process models that contain many interleaving situations.

Moreover, we have also compared the performance of ProDiGen with four of the state of the art process discovery algorithms, using non-parametric statistical tests. The selected algorithms are: (i) α^{++}-algorithm [41]; (ii) Heuristics Miner (HM) [40]; (iii) Genetic Miner (GM) [8]; and (iv) ILP [35]. These are well-known algorithms for process models discovery from event logs, and they are available in the ProM framework [36], a very complete and excellent tool for process mining and analysis.

5.1. Logs

The algorithms have been tested with 39 process models, and a total of 111 different logs[4]. From those models, 18 out of 39 were used to generate 90 synthetic logs with different degrees of noise. The rest of the logs —21 out of 111— come directly from [8] and [4] with their corresponding original models.

5.1.1. Balanced logs

We have conducted an experiment with 18 different process models with increasing degrees of complexity. Table 2a summarizes the structural complexity of these models ranging from 5 to 16 tasks that contain sequences, choices, parallelism, loops and non-free-choice constructs. For each of these models, a *synthetic* log was randomly generated with all the possible paths represented in the model. Tab. 2a also shows the characteristics of the logs, where column *#traces* indicates the number of traces and the column *#events* the number of total activities in the event log.

Afterwards, each noise-free log was used to generate another four logs with 1%, 5%, 10% and 20% of noise. Four different types of noise were used [22]: (i) *missing head*; (ii) *missing body*; (iii) *missing tail*; and (iv) *swap tasks*. We followed the same strategy as [8] to generate the noisy traces. Assuming that each trace is defined as $\sigma = t_1 \cdots t_n$, these noise types behave as follows. The first three types, respectively, randomly remove sub-traces of events from the head, body and tail. The head goes from t_1 to $t_{n/3}$, the body goes from $t_{(n/3)+1}$ to $t_{2n/3}$ and the tail goes from $t_{(2n/3)+1}$ to t_n. Swap noise interchanges two random chosen events.

To incorporate noise, the traces of the original noise-free logs were randomly selected and then one of the four noise types was applied —each one with an equal probability of 0.25. This combination of noise types is called *mixed noise*. We focus our tests on this noise type, as it is the typical noise in real logs. Note that the number of traces of each log is the same after applying the noise, but the number of events may change —some type of noise may remove events from cases, reducing the number of total events in the log.

[4]The reader can found all the logs and models used in our experimentation in `http://tec.citius.usc.es/SoftLearn/ProDiGen_ins.html`.

Model	#Tasks	Sequence	Choice	Parallelism	Length-One Loop	Length-Two Loop	Arbitrary Loop	Structured Loop	Non-local NFC	Local NFC	Invisible tasks	#traces	#events
	Activity structures											Log content	
Caminatas	12	✓	✓	✓								700	4,200
A8	7	✓	✓	✓								300	1,200
D2	6	✓		✓								300	1,200
Ml1lSkip [8]	6	✓	✓	✓	✓							500	4,757
Ma5 [8]	7	✓	✓	✓	✓							300	2,178
Ml2l [8]	6	✓	✓			✓						300	4,668
MDriverLL [8]	11	✓	✓	✓				✓	✓		✓	700	13,303
allLoops	5	✓	✓		✓	✓		✓			✓	300	1,035
l2la	6	✓	✓	✓			✓					300	2,264
Ma7 [8]	9	✓	✓	✓								500	2,427
Herbst6p37 [8]	16	✓		✓								700	12,600
MexampleL [8]	8	✓	✓	✓								300	1,645
Ma6nfc [8]	8	✓	✓							✓		300	2,006
MParallel5 [8]	10	✓		✓								700	12,600
NC	7	✓	✓	✓						✓		300	1,704
L2LP	7	✓	✓	✓	✓	✓		✓			✓	300	5,476
NCB	7	✓	✓	✓			✓			✓		300	2,950
DWS [14]	12	✓	✓	✓							✓	500	4,033

(a) *Balanced logs*.

Model	#Tasks	Sequence	Choice	Parallelism	Length-One Loop	Length-Two Loop	Arbitrary Loop	Structured Loop	Invisible tasks	Unbalanced AND-join/split	#traces	#events
		Activity structures									Log content	
g2 [8]	22	✓	✓	✓		✓	✓		✓		300	4501
g3 [8]	29	✓	✓	✓			✓	✓	✓		300	14599
g4 [8]	29	✓	✓	✓		✓				✓	300	5975
g5 [8]	20	✓	✓	✓				✓	✓		300	6172
g6 [8]	23	✓	✓	✓		✓			✓		300	5419
g7 [8]	29	✓	✓	✓			✓		✓		300	14451
g8 [8]	30	✓	✓	✓		✓	✓		✓	✓	300	5133
g9 [8]	26	✓	✓	✓		✓	✓		✓		300	5679
g10 [8]	23	✓	✓	✓				✓	✓		300	4117
g12 [8]	26	✓	✓	✓		✓		✓	✓		300	4841
g13 [8]	22	✓	✓	✓	✓	✓			✓	✓	300	5007
g14 [8]	24	✓	✓	✓			✓		✓	✓	300	11340
g15 [8]	25	✓	✓		✓	✓			✓		300	3978
g19 [8]	23	✓	✓	✓		✓			✓	✓	300	4107
g20 [8]	21	✓	✓		✓	✓		✓	✓		300	6193
g21 [8]	22	✓	✓					✓	✓		300	3882
g22 [8]	24	✓	✓	✓			✓		✓	✓	300	3095
g23 [8]	25	✓	✓	✓		✓				✓	300	9654
g24 [8]	21	✓	✓	✓				✓	✓	✓	300	4130
g25 [8]	20	✓	✓	✓	✓				✓		300	6312
EMT[4]	7	✓	✓	✓					✓		100	790

(b) *Unbalanced logs*.

Table 2: Process models used in the experimentation

5.1.2. Unbalanced logs

The second type of logs consisted in 21 more complex case scenarios without noise. These models and logs[5] are summarized in Tab. 2b. Some of the models used in this experimentation contain *unbalanced AND-split/join points*, i.e., there is not a one-to-one relation between the AND-split points and the AND-join points. Moreover, all the logs are imbalanced, i.e., they contain traces with very different frequencies, as it is unrealistic to assume that, from a model with many interleaving situations, all the possible paths are equally executed. Hence, with this experiment, we can check whether an algorithm overfits or underfits the data due to the unbalanced frequencies of the traces in the log.

5.2. Metrics

The performance of the process discovery algorithms over the different logs has been measured with two different sets of metrics: (i) metrics based on the original model; and (ii) metrics based on the event log.

5.2.1. Metrics based on the original model

We use the metrics defined in [8] to compare the original and mined models. *Behavioral precision* (B_p) and *Behavioral recall* (B_r) detect, respectively, if the mined model can process traces that cannot be parsed by the original model, and if the original model can parse traces that cannot be processed in the mined model. On the other hand, *Structural precision* (S_p) and *Structural recall* (S_r) checks, respectively, if there are causality relations of the mined

[5] In this experiment, both the process models and the logs were taken from other papers [4, 8].

model that are not defined in the original model, and if there are causality relations of the original model that are not defined in the mined model.

The mined model is as precise as the original one if $B_p = 1$ and $B_r = 1$: the closer the values of B_p and B_r to 1, the higher the similarity between the original and the mined models. Although two models could be equal from the *behavioral point of view*, their structures may be different. S_p and S_r measure the similarity from the *structural point of view*. When the original model has connections that do not appear in the mined model, S_r will take a value smaller than 1, and, in the same way, when the mined model has connections that do not appear in the original model, S_p will take a value lower than 1.

5.2.2. Metrics based on the log

Additionally to the four previously described metrics, we have also used three metrics to measure the completeness, precision and simplicity taking into account the information of the log. To measure the completeness (C), we use the *proper completion* metric [26], which is the fraction of properly completed process instances. *Proper completion* takes a value of 1 if the mined model can process all the traces without having missing tokens or tokens left behind. Also, the precision (P) is evaluated as follows:

$$P = 1 - max\{0, P'_o - P'_m\} \tag{4}$$

where P'_o and P'_m are, respectively, the precision of the original model and the precision of the mined model, both calculated with the *alignment precision* defined in [30]. As the original model is the optimal solution, we use it to normalize the precision. Therefore, P will be equal to 1 if the mined model has a precision (P'_m) equal or higher than the original model (P'_o). When the precision of the mined model is worse than that of the original model, P will take a value under 1 —the lower the precision of the mined model, the closer the value of P to 0. Finally, for the simplicity (S) we use:

$$S = \frac{1}{1 + max\{0, S'_m - S'_o\}} \tag{5}$$

where S'_m and S'_o are, respectively, the simplicity of the mined model and the simplicity of the original model, both calculated with the *weighted P/T average arc degree* defined in [27] —the higher the value of S' the lower the simplicity. As explained with the precision, we use the original model —which is the optimal model— to normalize the simplicity. S takes a value of 1 if the simplicity of the mined model is equal or higher than that of the original model, i.e., $S'_m \leq S'_o$. If the simplicity of the mined model is worse than that of the original model ($S'_m > S'_o$), S will take a value under 1 —the worse the simplicity of the mined model, the closer the value of S to 0. To measure the metrics C, P' and S' we have used the tool CoBeFra [38].

5.3. Settings

The settings of the different algorithms were mostly kept to the default options of ProM 5.2 and ProM 6.3. However, some modifications were made for Heuristics Miner and Genetic Miner to keep the configurations specified by the authors of the algorithms. To be more specific, the settings used are:

- α^{++}*-algorithm*: the algorithm has no settings (ProM 5.2).
- *Heuristics Miner*. We used the default settings established in ProM 6.3: relative-to-best = 0.05, dependency = 0.9, length-one-loops = 0.9, lenght-two-loops = 0.9, long distance = 0.9. Additionally, *mine long distance dependencies* was enabled. These parameters were set for both the balanced and unbalanced logs.
- *Genetic Miner*. We set the values of the parameters equal to those established in [8], as 29 out of the 39 models used in this experimentation were also tested in [8]:
  - Settings for the balanced logs: iterations = 1,000, population size = 100, elitism rate = 0.2, crossover probability 0.8, mutation probability = 0.2 (per task), elitism rate = 0.02, selection type = tournament 5, extra behavior punishment = 0.025 and prune threshold = 0.1%.
  - Settings for the unbalanced logs: the same as for the balanced logs, except iterations = 5,000, population size = 10, elitism rate = 0.2, selection type = binary and prune threshold = 0%.
- *ProDiGen*:
  - Settings for the balanced logs: iterations = 1,000, population size = 100, crossover probability 0.8, initialTimesRun = 35, maxRestarts = 5, prune threshold = 0.1. For the pre-processing parameter $\xi = 2$, i.e., two standard deviations.
  - Settings for the unbalanced logs: the same as for the balanced logs, except prune threshold = 0%.
- *ILP*: we selected the default settings defined in ProM 6.3: ILP Solver = Java-ILP & LPSolve 5.5, ILP Variant = Petri net (Empty net after completion), number of places = Per Causal Dependency and *search for separate initial places* was enabled.

For ProDiGen, we kept exactly the same settings as Genetic Miner, except for the new parameters related with the reinitialization process, the pre-processing parameter ξ, and the mutation probability that in ProDiGen is always 1, as explained in Sec 4.1.4. The values of maxRestarts and initialTimesRun do not affect the results of ProDiGen — provided that they are not drastically reduced; increasing those values only augments the execution time of ProDiGen. Moreover, we did not modify ProDiGen parameters when dealing with more complex logs, contrary to Genetic Miner which requires to increase the maximum number of iterations to converge when the complexity of the log increases. Additionally, we did not apply any post-pruning process for the unbalanced logs for both ProDiGen and Genetic Miner, as these are noise-free logs.

5.4. Results on balanced logs

Tables 3-5 present the results of the algorithms with the balanced logs for the different percentages of noise. The rows show the values of the 4 metrics based on the original model and the 3 metrics based on the log. As both

			Logs with 0% of noise																		Logs with 1% of noise																	
			Caminatas	A8	D2	MI1Skip	Ma5	MI21	MDriverLL	allLoops	Ma7	I2la	MexampleL	Herbst6p37	Ma6nfc	MParallel5	NC	L2LP	NCB	DWS	Caminatas	A8	D2	MI1Skip	Ma5	MI21	MDriverLL	allLoops	Ma7	I2la	MexampleL	Herbst6p37	Ma6nfc	MParallel5	NC	L2LP	NCB	DWS
ProDiGen	Model metrics	Bp	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		Br	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		Sp	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		Sr	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
GM	Model metrics	Bp	0.84	1.0	1.0	1.0	0.96	1.0	0.9	0.99	0.95	1.0	0.94	1.0	0.98	0.74	1.0	1.0	1.0	1.0	0.68	1.0	0.95	0.99	0.97	1.0	0.92	0.99	0.82	1.0	1.0	1.0	0.98	0.8	1.0	1.0	1.0	0.86
		Br	0.98	1.0	1.0	1.0	1.0	1.0	1.0	0.99	0.99	1.0	0.97	1.0	1.0	0.97	1.0	1.0	1.0	1.0	0.99	0.99	1.0	0.96	0.99	0.99	1.0	1.0	0.99	1.0	0.98	1.0	1.0	1.0	0.81			
		Sp	0.85	1.0	1.0	1.0	1.0	1.0	0.93	0.98	0.94	1.0	0.86	1.0	0.96	0.08	1.0	1.0	1.0	1.0	0.79	0.8	1.0	1.0	0.91	0.88	0.94	1.0	0.75	1.0	1.0	0.95	0.96	0.85	1.0	1.0	1.0	0.83
		Sr	0.73	1.0	1.0	1.0	0.91	1.0	0.84	0.96	0.97	1.0	0.88	1.0	0.99	0.98	1.0	1.0	1.0	1.0	0.67	1.0	0.87	0.92	0.91	1.0	0.85	0.96	1.0	1.0	1.0	1.0	0.99	0.93	1.0	1.0	1.0	0.95
	Log metrics	C	0.36	1.0	1.0	1.0	1.0	1.0	1.0	0.48	0.15	1.0	0.52	1.0	1.0	0.81	1.0	1.0	1.0	1.0	0.23	1.0	1.0	1.0	1.0	1.0	1.0	0.49	1.0	1.0	1.0	1.0	1.0	0.14	1.0	1.0	1.0	0.82
		P	0.81	1.0	1.0	1.0	0.97	1.0	0.93	0.92	0.68	1.0	0.97	1.0	0.81	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.63	0.96	0.98	0.88	0.88	0.82	0.76	1.0	1.0	0.94	0.92	0.75	1.0	1.0	1.0	0.93
		S	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.93	1.0	1.0	1.0	1.0	0.96	1.0	1.0	1.0	1.0	1.0	1.0	0.72	0.88	0.73	1.0	0.85	1.0	0.9	1.0	1.0	1.0	0.92	0.93	0.67	1.0	1.0	1.0	0.88
HM	Model metrics	Bp	1.0	1.0	1.0	0.97	1.0	1.0	1.0	0.96	1.0	1.0	1.0	1.0	0.91	1.0	0.44	0.82	0.88	0.89	1.0	1.0	1.0	0.97	1.0	1.0	0.93	0.96	1.0	1.0	1.0	1.0	0.9	1.0	0.87	0.82	0.88	0.89
		Br	1.0	1.0	1.0	0.95	1.0	1.0	1.0	0.92	1.0	1.0	1.0	1.0	1.0	1.0	0.82	0.93	1.0	0.94	1.0	1.0	1.0	0.92	1.0	1.0	1.0	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.93	1.0	0.94
		Sp	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.83		1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.81	
		Sr	1.0	1.0	1.0	0.82	1.0	1.0	1.0	0.83	1.0	1.0	1.0	1.0	0.9	1.0	0.91	0.9	0.88	1.0	1.0	1.0	1.0	0.76	1.0	1.0	0.9	0.83	1.0	1.0	1.0	1.0	0.91	1.0	0.91	0.9	0.88	1.0
	Log metrics	C	1.0	1.0	1.0	0.84	1.0	1.0	1.0	0.73	1.0	1.0	1.0	1.0	0.69	1.0	0.84	0.24	0.0	0.51	1.0	1.0	1.0	0.84	1.0	1.0	1.0	0.73	1.0	1.0	1.0	1.0	0.69	1.0	0.84	0.24	0.0	0.51
		P	1.0	1.0	1.0	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.88	1.0	0.83	1.0	0.14	1.0	1.0	1.0	1.0	0.89	1.0	1.0	0.91	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.87	1.0	0,14	1.0
		S	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,88	1.0	0,98	1.0	1.0	1.0	0,98	1.0	1.0	1.0	0,79	1.0	1.0	1.0	1.0	1.0	1.0	0,89	1.0	0,98	
α^{++}	Model metrics	Bp	1.0	1.0	1.0	1.0	1.0	1.0	0.87	0.77	1.0	1.0	1.0	1.0	1.0	1.0	0.59	0.88	0.75		0.76	0.82	1.0	0.73	0.84	0.76	0.77	0.94	0.83	0.73	0.4	0.41	0.6	0.87	0.76	0.77	0.88	0.46
		Br	1.0	1.0	1.0	1.0	1.0	1.0	0.89	0.83	1.0	1.0	1.0	1.0	1.0	1.0	0.72	1.0	0.63		0.87	0.87	1.0	0.63	0.82	0.69	0.75	0.83	0.85	0.6	0.7	0.52	0.78	0.99	0.86	0.56	1.0	0.45
		Sp	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.72	1.0	0.81		0.93	0.77	1.0	0.81	0.83	1.0	0.79	0.81	0.83	0.69	0.69	0.46	0.68	0.68	0.84	1.0	1.0	0.74
		Sr	1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.88	1.0		1.0	0.87	1.0	0.69	0.83	0.75	0.9	0.75	0.83	0.81	0.75	0.61	0.91	0.86	0.91	0.6	0.88	0.9
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	0.63	0.25	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.5	1.0	1.0	1.0	1.0	0.0	0.25	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,97	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,44	0,14	0,23	0.0	1.0	1.0	0,19	0,96	0,83	0,11	0,96	0.0	0,05	0,05	0,05	0.0	0,01	0.0	0,45	0,14	0,23
		S	1.0	1.0	1.0	1.0	1.0	1.0	0,81	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,52	1.0	0,71	0,83	0,86	1.0	0,75	0,63	1.0	0,19	0,72	1.0	0,75	0,8	0,72	0,49	0,75	0,83	0,52	0,83	0,35
ILP	Model metrics	Bp	1.0	1.0	1.0	1.0	1.0	1.0	0.86	0.81	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.63	1.0	0.82	0.3	0.39	0.64	0.58	0.84	0.63	0.32	0.31	0.42	0.48	0.47	0.33	0.47	0.3	0.52	0.62	0.63	0.42
		Br	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.97	0.68	0.56	0.83	0.75	0.89	0.9	0.68	0.53	0.69	0.7	0.75	0.89	0.78	0.74	0.84	0.9	0.99	0.56
		Sp	1.0	1.0	1.0	1.0	1.0	1.0	0.86	0.54	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.66	1.0	0.66	0.28	0.53	0.63	0.69	0.75	0.76	0.32	0.53	0.43	0.34	0.41	0.87	0.49	0.45	0.45	0.66	0.66	0.29
		Sr	1.0	1.0	1.0	1.0	1.0	1.0	0.93	0.71	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.83	1.0	1.0	0.87	1.0	1.0	1.0	0.95	0.92	0.88	0.96	1.0	0.98	1.0	1.0	1.0	1.0
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	0,93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,91	1.0	0,95		0,47	0,5	0,72	0,82	0,96	0,7	0,64	0,79	0,57	0,05	0,79	0,6	0,62	0,74	0,63	0,92	0,67	0,74
		S	1.0	1.0	1.0	1.0	1.0	1.0	0,99	0,61	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,9	1.0	0,63	0,3	0,64	0,59	0,56	0,63	0,88	0,22	0,34	0,42	0,34	0,53	0,56	0,49	0,38	0,39	0,9	0,88	0,2

Table 3: Results on the balanced logs with a 0% and 1% of noise.

ProDiGen and Genetic Miner are non-deterministic algorithms, the results shown in the tables are the average results of 10 executions.

For the noise-free logs (Tab. 3) ProDiGen obtains the original model in all the cases, as the values for both the model and log metrics are 1, proving that it can handle all the main workflow constructs at once. On the other hand, Genetic Miner is not able to get the original model for several logs. For instance, for logs *Ma6nfc* and *Ma5*, Genetic Miner always gets a complete model but, on average —ten executions for each log—, the mined model is different from the original model. This is closely related to the simplicity of the mined model as there are needless relations that increase the complexity of the model. Heuristics Miner shows a great performance, but it has difficulties to mine some non-free-choice constructs and short loops. ILP has problems tackling invisible tasks and, finally, the α^{++}-algorithm fails when dealing with local non-free-choices and invisible tasks.

For the noisy logs (Tables 3-5) we have used the same metrics. However, to evaluate the seven metrics we did not consider the noisy traces, i.e., a completeness of 1 means that the model is able to process all the noise-free traces of the noisy log. ProDiGen shows again a great performance, getting the original models for all the logs with a noise percentage up to 5%, and all the original models but one for a percentage of 10%. For the logs with a percentage of the 20% of noise the performance decreases, although ProDiGen finds 6 out of 18 of the original models and is the best algorithm. These results are related with the fitness definition of ProDiGen, that tries to better fit the behavior

of the log. This proves that the pre-processing —filtering the less used traces— and post-processing —pruning the infrequent arcs— improves the results of ProDiGen.

			Logs with 5% of noise																	Logs with 10% of noise																
			Caminatas	A8-	D2	MI1ISkip	Ma5	MI21	MDriverLicensed	allLoops	Ma7	l2la	MexampleLog	MherbstFig6p37	Ma6nfc	MParallel5	NC	L2LP	NCB	DWS	Caminatas	A8-	D2	MI1ISkip	Ma5	MI21	MDriverLicensed	allLoops	Ma7	l2la	MexampleLog	MherbstFig6p37	Ma6nfc	MParallel5	NC	L2LP	NCB	DWS
ProDiGen	Model metrics	B_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		B_r	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.73	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S_r	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.78	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
GM	Model metrics	B_p	0.8	0.91	0.94	0.99	0.84	1.0	0.88	0.99	0.94	0.94	0.87	0.79	1.0	0.86	0.85	0.95	1.0	0.86	0.92	0.83	0.8	1.0	0.87	1.0	0.92	0.99	0.77	0.87	0.99	0.92	0.88	0.86	0.89	1.0	0.82	0.84
		B_r	0.99	0.99	0.99	0.96	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	0.99	0.99	1.0	1.0	0.81	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1.0	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.83
		S_p	0.66	0.77	0.77	0.92	0.66	0.88	0.8	1.0	0.8	0.9	0.68	0.76	0.92	0.66	0.69	0.9	1.0	0.83	0.84	0.6	0.75	0.92	0.9	0.88	0.9	0.98	0.57	0.75	0.92	0.95	0.83	0.66	0.91	0.9	0.72	0.79
		S_r	0.5	0.87	0.87	0.92	0.66	1.0	0.76	0.96	1.0	0.9	0.91	0.95	1.0	0.66	0.75	0.9	1.0	0.95	0.78	0.75	0.75	1.0	0.83	1.0	0.85	0.96	0.66	0.81	1.0	0.95	0.83	0.66	0.91	1.0	0.88	0.95
	Log metrics	C	0.22	1.0	1.0	1.0	0.69	1.0	0.38	1.0	0.86	1.0	1.0	0.76	1.0	1.0	0.55	0.77	1.0	0.27	0.23	0.5	1.0	1.0	1.0	1.0	1.0	0.61	0.59	1.0	0.53	1.0	1.0	1.0	0.17	1.0	1.0	0.57
		P	0.68	0.63	0.86	0.86	0.75	0.81	0.64	0.75	0.77	0.9	0.58	0.65	0.91	0.47	0.62	0.99	1.0	0.82	0.78	0.55	0.45	0.86	0.6	0.58	0.73	0.65	0.51	0.48	0.71	0.79	0.42	0.58	0.75	1.0	0.7	0.76
		S	1.0	0.69	0.88	0.64	0.92	0.85	0.94	0.85	0.76	1.0	0.85	0.81	0.76	0.78	0.87	1.0	1.0	0.89	1.0	0.69	0.91	0.73	0.87	0.71	0.95	0.79	0.85	0.84	0.77	0.88	0.74	0.77	0.83	0.83	0.93	0.84
HM	Model metrics	B_p	0.96	1.0	1.0	0.97	0.92	0.91	0.92	0.93	1.0	1.0	1.0	1.0	0.9	1.0	0.87	0.77	0.88	0.77	0.96	1.0	0.91	1.0	0.92	0.86	0.92	0.93	1.0	0.94	1.0	1.0	0.9	1.0	0.87	0.96	0.88	0.86
		B_r	1.0	1.0	1.0	0.92	0.79	0.99	0.95	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.95	1.0	0.94	1.0	1.0	1.0	0.89	0.79	0.99	0.95	0.91	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.81
		S_p	1.0	1.0	1.0	1.0	1.0	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	1.0	0.83	1.0	1.0	1.0	1.0	1.0	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.83
		S_r	0.92	1.0	1.0	0.76	0.91	1.0	0.9	0.83	1.0	1.0	1.0	1.0	0.91	1.0	0.91	0.8	0.88	1.0	0.93	1.0	0.75	0.84	1.0	1.0	0.9	0.83	1.0	0.9	1.0	1.0	0.91	1.0	0.91	0.9	0.88	0.95
	Log metrics	C	0.66	1.0	1.0	0.0	0.0	1.0	0.63	0.26	1.0	1.0	1.0	1.0	0.69	1.0	0.85	0.77	0.0	0.0	1.0	1.0	1.0	0.9	0.0	1.0	0.62	0.34	1.0	0.3	1.0	1.0	0.69	1.0	0.84	0.0	0.0	0.0
		P	0,92	1.0	1.0	0,19	0,1	0,72	0,98	1.0	1.0	1.0	1.0	1.0	0,9	1.0	0,87	0,99	0,14	0,23	1.0	1.0	0,91	0,19	0,1	0,71	0,99	0,96	1.0	0,93	1.0	1.0	0,9	1.0	0,87	0,45	0,14	0,23
		S	1.0	1.0	1.0	0,91	1.0	0,91	1.0	0,79	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,92	1.0	1.0	1.0	0,88	1.0	0,83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,96	1.0	0,91
α^{++}	Model metrics	B_p	0.73	0.81	0.69	0.34	0.75	0.63	0.38	0.68	0.83	0.72	0.55	0.38	0.8	0.45	0.75	0.76	0.23	0.57	0.85	0.84	0.84	0.86	0.64	0.61	0.35	0.74	0.84	0.65	0.63	0.33	0.54	0.63	0.62	0.68	0.32	0.43
		B_r	0.69	0.99	0.83	0.32	0.88	0.68	0.49	0.67	0.97	0.65	0.6	0.47	0.88	0.54	0.99	0.56	0.24	0.66	0.95	0.99	0.99	0.66	0.64	0.65	0.45	0.77	0.99	0.68	0.69	0.41	0.6	0.75	0.97	0.52	0.36	0.56
		S_p	0.62	0.41	0.57	0.31	0.46	0.3	0.41	0.5	0.47	0.43	0.36	0.4	0.55	0.37	0.5	0.6	0.5	0.48	0.61	0.46	0.71	0.46	0.3	0.35	0.35	0.41	0.44	0.58	0.37	0.36	0.31	0.29	0.25	0.45	0.66	0.31
		S_r	0.82	0.62	0.5	0.38	0.5	0.37	0.57	0.58	0.66	0.63	0.66	0.56	0.91	0.6	0.83	0.6	0.66	0.72	0.75	0.75	0.62	0.53	0.5	0.62	0.61	0.41	0.66	0.63	0.75	0.45	0.58	0.66	0.33	0.5	0.44	0.42
	Log metrics	C	0.0	0.49	1.0	0.0	1.0	0.0	0.0	0.0	0.86	0.0	0.0	0.0	0.69	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.57	0.82	0.0	0.0	0.0	0.0	0.84	1.0	0.0	0.0
		P	0,01	0,68	0,94	0,19	0,84	0,16	0,11	0,16	0,84	0,05	0,05	1.0	0.0	0,01	0.0	0,45	0,14	0,23	0.0	0,84	0,84	0,19	0,1	0,15	0,11	0,15	0,76	0,75	0,05	0,99	0.0	0,01	0,75	1.0	0,14	0,23
		S	0,36	0,64	1.0	0,28	0,81	0,36	0,15	0,26	0,68	0,28	0,36	1.0	0,55	0,55	0,4	0,39	0,3	0,25	0,52	0,6	1.0	0,3	0,32	0,36	0,19	0,34	0,64	0,41	0,35	1.0	0,37	0,37	0,59	0,81	0,99	0,22
ILP	Model metrics	B_p	0.13	0.15	0.24	0.48	0.31	0.14	0.07	0.27	0.05	0.14	0.18	0.1	0.24	0.34	0.28	0.36	0.26	0.25	0.16	0.25	0.2	0.33	0.32	0.2	0.12	0.28	0.21	0.12	0.15	0.05	0.13	0.17	0.1	0.41	0.24	0.15
		B_r	0.3	0.48	0.83	0.33	0.62	0.42	0.21	0.81	0.17	0.44	0.36	0.45	0.59	0.64	0.4	0.69	0.75	0.49	0.25	0.72	0.58	0.47	0.64	0.5	0.19	0.64	0.38	0.42	0.29	0.15	0.43	0.29	0.32	0.72	0.47	0.24
		S_p	0.23	0.23	0.32	0.26	0.26	0.17	0.16	0.33	0.23	0.21	0.13	0.09	0.16	0.16	0.23	0.3	0.27	0.13	0.24	0.19	0.34	0.27	0.18	0.18	0.15	0.36	0.19	0.25	0.16	0.06	0.16	0.13	0.18	0.28	0.28	0.12
		S_r	1.0	1.0	0.8	0.95	0.93	1.0	1.0	1.0	1.0	0.92	0.92	0.94	1.0	0.78	1.0	1.0	0.91	0.96	1.0	0.83	0.8	1.0	0.87	1.0	1.0	0.95	1.0	1.0	1.0	1.0	1.0	1.0	0.93	1.0	1.0	0.92
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0,32	0,4	0,41	0,49	0,1	0,42	0,28	0,6	0,57	0,34	0,43	0,05	0,41	0,42	0,47	0,77	0,44	0,58	0,35	0,4	0,43	0,52	0,1	0,4	0,27	0,59	0,42	0,41	0,41	0,05	0,36	0,3	0,31	0,8	0,41	0,5
		S	0,14	0,27	0,74	0,16	0,27	0,24	0,1	0,52	0,26	0,21	0,19	0,13	0,18	0,19	0,22	0,38	0,33	0,13	0,17	0,31	0,44	0,22	0,29	0,26	0,1	0,4	0,23	0,24	0,21	0,07	0,2	0,12	0,19	0,35	0,28	0,15

Table 4: Results on the balanced logs with a 5% and 10% of noise.

With respect to the results of the other algorithms, Heuristics Miner obtains the same results as for the noise-free logs with very low noise levels (1%), but, the higher the frequency of noise, the worse the results. Heuristics Miner is known to be good retrieving the main behavior of the log, hence, since these logs contain a mixture of all noise types, some noise dependencies can be as frequent as the correct ones, resulting in incorrect models —excluding those noise-free logs that the Heuristics Miner cannot correctly mine. Nevertheless, with high levels of noise, it obtains very competitive results. On the other hand, ILP usually gets a complete model, but it produces very complex and underfitted models because of its inability to handle infrequent behavior. Finally, α^{++} is not robust to noise and, therefore, almost all of its mined models are not even complete. In summary, ProDiGen correctly mines, i.e, finds the original model, the 85% (77 out of 90) of the cases. For the other algorithms, the percentage of correctly mined models was: GM in the 16% (16 out of 90), HM in the 35% (32 out of 90), α^{++} in the 15% (14 out of 90), and ILP in the 15% (14 out of 90).

We have compared the results of the algorithms by means of non-parametric statistical tests. We first applied the Friedman test [12] that computes the ranking of the results of the algorithms, and rejects the null hypothesis —which states that the results of the algorithms are equivalent— with a given confidence or significance level (α).

Then we applied the Holm's post-hoc test [18] for detecting significant differences among the results. However, as we are using 7 different metrics, the comparison must be done in a multi-objective way. In order to perform a fair comparison, we have used the criterion of Pareto dominance. We have applied the *fast-non-dominated-sort* [19] in order to rank the solutions of the algorithms for each log. With this method, a mined model a dominates other mined model b, i.e., $a \succ b$, if the model a is not worse than the model b in all the objectives —the 7 metrics— and better in at least one objective. Thereby, for each log, all the solutions in the first non-dominated front will have a rank equal to 1 —these are the solutions more similar to the original model and, therefore, the best ones—, the solutions in the second non-dominated front will have a rank equal to 2, and the process continues until all fronts are identified[6]. Therefore, to perform the non-parametric statistical tests, we first ranked all the solutions for each log based on the Pareto dominance and then we used these ranks as input for the Friedman test.

			Logs with 20% of noise																
			Caminatas A8	D2	M11Skip Ma5	M12l	MDriverLL allLoops	Ma7	l2la	MexampleL Herbst6p37	Ma6nlc	MParallel5 NC	L2LP	NCB	DWS-20				
ProDiGen	Model metrics	B_p	0.57	1.0	0.91	1.0	1.0	1.0	0.62	1.0	0.78	0.8	0.75	1.0	0.75	0.92	0.72	0.76	0.8	0.5
		B_r	0.99	1.0	1.0	1.0	1.0	1.0	0.99	1.0	0.99	0.99	0.99	1.0	0.99	0.99	0.99	0.99	0.97	0.99
		S_p	0.75	1.0	0.87	1.0	1.0	1.0	0.77	1.0	0.79	0.81	0.76	1.0	0.76	0.78	0.76	0.78	0.8	0.47
		S_r	0.85	1.0	0.87	1.0	1.0	1.0	0.89	1.0	1.0	1.0	0.83	1.0	0.83	0.84	0.75	0.85	1.0	0.95
	Log metrics	C	0.34	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.52	1.0	0.69	0.72	0.62	1.0	1.0	0.37
		P	0.63	1.0	0.95	1.0	1.0	0.15	1.0	1.0	1.0	1.0	0.95	1.0	0.68	0.8	0.7	0.76	0.89	0.98
		S	1.0	1.0	1.0	1.0	1.0	0.22	1.0	1.0	1.0	1.0	1.0	1.0	0.97	0.91	0.98	0.95	0.91	0.92
GM	Model metrics	B_p	0.56	0.67	0.66	0.85	0.82	1.0	0.44	0.9	0.67	0.77	0.73	0.59	0.65	0.83	0.57	0.95	0.88	0.78
		B_r	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.98	0.99	0.99	0.99	0.97	0.99	0.99	0.99	0.99	0.99	0.81
		S_p	0.56	0.46	0.6	0.69	0.72	0.88	0.52	0.88	0.47	0.69	0.64	0.44	0.61	0.52	0.5	0.9	0.8	0.6
		S_r	0.6	0.75	0.75	0.69	0.66	1.0	0.52	0.95	0.66	0.81	0.91	0.65	0.66	0.66	0.66	0.9	0.88	0.95
	Log metrics	C	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.3	1.0	0.16	1.0	1.0	0.54
		P	0,4	0,42	0,45	0,84	0,45	0,8	0,11	0,61	0,42	0,47	0,43	0,05	0,38	0,6	0,37	0,76	0,65	0,76
		S	1.0	0,64	0,91	0,77	0,75	0,85	1.0	0,58	0,81	0,76	0,65	0,63	0,68	0,88	0,84	0,68	0,88	0,84
HM	Model metrics	B_p	0.97	0.7	0.95	0.9	0.9	0.67	0.9	0.93	1.0	0.94	1.0	1.0	0.89	1.0	0.75	0.77	0.88	0.7
		B_r	1.0	0.85	1.0	0.8	0.89	0.91	0.92	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.95	1.0	0.6
		S_p	1.0	0.77	1.0	1.0	1.0	0.6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	1.0	0.76
		S_r	0.95	0.87	0.87	0.76	0.83	0.75	0.9	0.83	1.0	0.9	1.0	1.0	0.91	1.0	0.83	0.8	0.88	0.95
	Log metrics	C	0.34	0.0	1.0	0.0	0.65	0.0	0.63	0.58	1.0	0.3	1.0	1.0	0.69	1.0	0.0	0.0	0.0	0.0
		P	0,9	0.0	0,95	0,19	0,94	0,15	0,11	1.0	1.0	0,93	1.0	1.0	0,9	1.0	0.0	0,45	0,14	0,23
		S	1.0	0,9	1.0	0,75	1.0	0,81	1.0	0,93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,96	1.0	0,93
α^{++}	Model metrics	B_p	0.83	0.88	0.84	0.39	0.61	0.4	0.29	0.73	0.76	0.42	0.4	0.29	0.44	0.75	0.53	0.62	0.29	0.26
		B_r	0.91	0.99	0.99	0.55	0.56	0.49	0.4	0.82	0.87	0.57	0.75	0.42	0.76	0.99	0.89	0.54	0.4	0.24
		S_p	0.59	0.46	0.62	0.23	0.26	0.26	0.18	0.2	0.27	0.25	0.17	0.21	0.23	0.22	0.26	0.15	0.27	0.29
		S_r	0.67	0.75	0.62	0.23	0.5	0.5	0.28	0.16	0.41	0.27	0.33	0.3	0.41	0.46	0.33	0.2	0.33	0.42
	Log metrics	C	0.0	0.49	1.0	0.0	0.0	1.0	0.0	0.24	1.0	0.0	0.52	1.0	0.0	1.0	0.83	0.0	0.0	0.0
		P	0,01	0,72	0,84	0,19	0,09	0,69	0,11	0,75	0,65	0,05	0,64	1.0	0.0	0,76	0.0	0,45	0,14	0,23
		S	0,5	0,58	1.0	0,22	0,29	0,88	0,11	0,86	0,63	0,24	0,41	1.0	0,4	0,42	0,62	0,27	0,36	0,22
ILP	Model metrics	B_p	0.1	0.09	0.27	0.35	0.31	0.11	0.24	0.29	0.17	0.1	0.2	0.05	0.07	0.2	0.17	0.35	0.41	0.14
		B_r	0.19	0.23	0.83	0.46	0.57	0.27	0.19	0.63	0.34	0.33	0.31	0.06	0.31	0.35	0.49	0.75	0.84	0.21
		S_p	0.23	0.23	0.45	0.28	0.21	0.17	0.19	0.26	0.2	0.22	0.14	0.05	0.14	0.1	0.19	0.29	0.25	0.12
		S_r	1.0	0.64	0.8	1.0	1.0	1.0	0.93	0.7	0.97	1.0	1.0	1.0	0.77	0.76	1.0	1.0	1.0	0.96
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0,32	0,4	0.0	0,49	0,37	0,35	0,27	0,59	0,38	0,32	0,35	0,18	0,28	0,27	0,31	0,71	0,36	0,45
		S	0,11	0,2	0,77	0,22	0,18	0,16	0,11	0,5	0,23	0,18	0,16	0,05	0,18	0,14	0,27	0,33	0,26	0,1

Table 5: Results on the balanced logs with a 20% of noise.

Algorithm	Ranking
ProDiGen	1.63
HM	2.52
GM	3.27
ILP	3.5
α^{++}	4.1
Friedman p-value: 5.34E-11	

(a) *Friedman ranking*.

i	Comp.	z	p	α/i	Hypothesis
4	α^{++}	10.4	9.73E-26	0.013	Rejected
3	ILP	7.9	2.87E-15	0.017	Rejected
2	GM	6.95	3.57E-12	0.025	Rejected
1	HM	3.77	1.62E-4	0.05	Rejected

(b) *Holm post-hoc,* $\alpha = 0.05$.

Table 6: Non-parametric test for the balanced logs.

Tab. 6 summarizes the results of the tests. As can be seen, ProDiGen has the best ranking, whereas Heuristics Miner gets the second position, as it retrieves very competitive results when dealing with noisy logs, but it still has problems with non-free-choice constructs and some short loops. On the other hand, Genetic Miner has the third position, closely followed by ILP miner. In general Genetic Miner retrieves better solutions than ILP, as ILP cannot

[6]Note that there can be as many fronts as possible solutions. Hence, as we are comparing five algorithms, there can be a maximum of 5 possible fronts, being the solutions in the first front the best ones.

handle invisible tasks and infrequent behavior. Finally, α^{++}-algorithm has the worst ranking. This is due the inability of this algorithm to tackle invisible tasks and handle infrequent behavior in the log, giving as a result incomplete and very underfited models. The p-value of the Friedman test (Tab. 6a) is really low, indicating a high level of confidence. Furthermore, based on the results of the Friedman test, we performed a Holm's post-hoc test (Tab. 6b), starting with the initial hypothesis that all the tested algorithms are equal to ProDiGen. The test rejects the null hypothesis in all the cases for a confidence level of $\alpha = 0,05$ —the *p-value* of each algorithm has to be lower than α/i in order to reject the hypothesis. This means that ProDiGen outperforms all the other algorithms, and that the difference is statistically significant for that confidence level. We have repeated this test considering only the model metrics, and also taking into account only the log metrics —the results of the tests do not change.

5.5. Results on unbalanced logs

			Logs																				
			g2	g3	g4	g5	g6	g7	g8	g9	g10	g12	g13	g14	g15	g19	g20	g21	g22	g23	g24	g25	EMT
ProDiGen	Model metrics	B_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.96	1.0
		B_r	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99	1.0
		S_p	1.0	1.0	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.91	1.0
		S_r	1.0	1.0	0.97	1.0	1.0	1.0	0.94	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.98	0.91	1.0
	Log metrics	C	1.0	1.0	0.78	1.0	1.0	1.0	0.52	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.98	1.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S	1.0	1.0	1.0	1.0	1.0	1.0	0,93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,94	1.0
GM	Model metrics	B_p	1.0	0.61	0.78	1.0	1.0	1.0	0.84	0.96	0.99	1.0	0.98	0.61	0.8	0.98	1.0	1.0	0.97	0.57	0.83	0.81	1.0
		B_r	1.0	0.97	0.97	1.0	1.0	1.0	1.0	1.0	0.97	1.0	0.99	1.0	0.97	0.9	1.0	1.0	1.0	0.88	0.88	0.96	0.83
		S_p	1.0	0.81	0.81	1.0	1.0	1.0	1.0	0.97	0.9	1.0	0.95	0.95	0.88	0.95	1.0	1.0	0.85	0.76	0.75	0.76	0.85
		S_r	1.0	0.81	0.81	1.0	1.0	1.0	0.94	0.98	0.92	1.0	0.94	0.94	0.87	0.89	1.0	1.0	0.85	0.74	0.75	0.74	0.85
	Log metrics	C	1.0	0.31	0.59	1.0	1.0	1.0	0.26	0.48	0.48	1.0	0.75	1.0	0.15	0.2	1.0	1.0	0.43	0.2	0.72	0.41	0.3
		P	1.0	0,6	1.0	1.0	1.0	1.0	0,15	1.0	1.0	1.0	1.0	0,81	0,14	0,08	1.0	1.0	0,96	0,42	0,99	0,75	0,94
		S	1.0	1.0	0,97	1.0	1.0	1.0	0,72	0,96	0,88	1.0	0,95	0,99	0,88	0,95	1.0	1.0	1.0	0,88	1.0	0,82	1.0
HM	Model metrics	B_p	1.0	1.0	0.94	1.0	0.9	0.97	0.87	1.0	0.96	1.0	1.0	0.97	0.96	0.97	1.0	1.0	0.99	0.6	0.92	0.76	0.81
		B_r	1.0	0.98	0.92	1.0	0.98	0.97	0.99	0.98	0.95	1.0	1.0	0.97	0.98	1.0	1.0	1.0	0.99	1.0	0.88	0.94	0.96
		S_p	1.0	0.97	0.96	1.0	0.93	0.97	0.95	1.0	0.96	1.0	1.0	0.96	1.0	1.0	1.0	1.0	0.97	0.91	0.89	0.85	0.76
		S_r	1.0	0.97	0.86	1.0	0.97	1.0	0.86	1.0	0.96	1.0	1.0	0.92	0.86	0.9	1.0	1.0	0.91	0.94	0.81	0.85	0.74
	Log metrics	C	1.0	1.0	0.78	1.0	0.66	1.0	0.52	0.74	0.78	1.0	1.0	0.91	0.87	0.85	1.0	1.0	0.9	0.0	0.93	0.23	0.37
		P	1.0	1.0	1.0	1.0	0,99	1.0	1.0	1.0	1.0	1.0	1.0	0,99	1.0	1.0	1.0	1.0	1.0	0,42	0,97	0,97	0,98
		S	1.0	1.0	1.0	1.0	0,99	0,98	0,93	0,96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,93	1.0	0,94	1.0
α^{++}	Model metrics	B_p	0.8	0.81	0.95	1.0	0.88	0.92	0.87	0.96	0.94	0.95	0.93	0.58	0.85	0.9	0.64	0.84	0.93	0.71	0.81	0.94	0.99
		B_r	0.85	0.91	0.94	1.0	0.99	0.9	0.94	0.94	0.9	1.0	0.94	0.92	0.85	0.94	0.94	0.89	0.85	0.67	0.7	0.92	0.93
		S_p	0.89	0.94	0.98	1.0	0.79	0.98	0.83	0.98	0.81	0.96	0.88	0.86	0.9	0.9	0.92	0.93	0.82	0.66	0.85	0.91	1.0
		S_r	1.0	0.98	0.97	1.0	1.0	1.0	0.94	1.0	1.0	1.0	0.97	1.0	0.97	1.0	0.87	1.0	0.97	1.0	1.0	0.98	0.91
	Log metrics	C	0.33	0.0	1.0	1.0	0.45	0.0	0.35	0.48	0.563	1.0	0.48	0.0	0.05	0.25	0.46	0.68	0.43	0.0	0.0	0.97	0.89
		P	0,96	0,18	0,97	1.0	1.0	0,12	1.0	1.0	1.0	0,97	1.0	0,82	0,14	0,98	0,86	0,09	1.0	0,42	0,11	0,26	1.0
		S	0,78	0,79	1.0	1.0	0,76	0,93	0,74	0,79	0,76	0,99	0,79	0,79	0,97	0,75	1.0	0,83	0,32	0,34	0,42	0,89	1.0
ILP	Model metrics	B_p	0.9	0.89	1.0	1.0	0.96	0.96	0.9	0.94	0.92	0.95	0.93	0.85	0.87	0.91	0.72	0.87	0.9	0.58	0.78	0.66	0.98
		B_r	1.0	0.98	0.99	1.0	0.99	0.99	0.99	0.99	0.97	1.0	0.99	0.99	1.0	0.99	0.95	1.0	0.99	0.88	0.94	0.96	1.0
		S_p	0.83	0.85	0.98	1.0	0.78	0.94	0.76	0.89	0.73	0.96	0.78	0.78	0.67	0.85	0.85	0.92	0.72	0.5	0.77	0.64	0.91
		S_r	1.0	0.98	0.97	1.0	1.0	1.0	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.97	1.0	1.0	0.98	1.0
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0,97	0,97	1.0	1.0	0,99	1.0	0,98	0,98	0,95	0,97	0,07	0,95	0,96	0,98	0,96	0,96	0,93	0,83	0,89	0,8	1.0
		S	0,93	0,92	0,96	1.0	0,74	0,93	0,66	0,9	0,68	0,99	0,97	0,69	0,76	0,79	0,79	0,97	0,52	0,28	0,7	0,59	0,93

(a) *Results on the unbalanced logs*.

Algorithm	Ranking
ProDiGen	1.55
HM	3.17
GM	3.31
ILP	3.43
α^{++}	3.55
Friedman p-value: 3.58E-7	

(b) *Friedman ranking*.

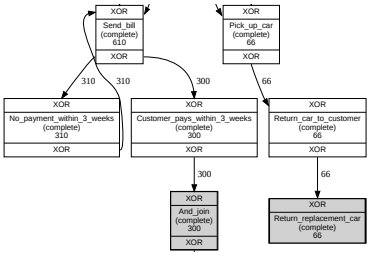
i	Comp.	z	p	α/i	Hypothesis
4	α^{++}	4.1	4.15E-5	0.013	Rejected
3	ILP	3.85	1.15E-4	0.017	Rejected
2	GM	3.61	3.05E-4	0.025	Rejected
1	HM	3.31	9.06E-4	0.05	Rejected

(c) *Holm post-hoc,* $\alpha = 0.05$.

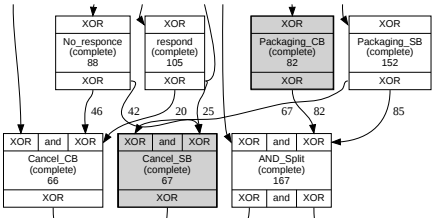
Table 7: Results and non-parametric test for the unbalanced logs.

Table 7a shows the results on the 21 unbalanced logs. ProDiGen mines the original model in 17 of the logs —the values of the four model metrics are 1—, while in the other 4 logs the mined model is very similar to the original one. The difficulties in these 4 logs arise when (i) mining logs with parallel constructs with more than two branches and with two or more tasks in each branch, and (ii) when mining logs that came from models with unbalanced AND-join/split points. These type of patterns are even more difficult to mine considering that not all the possible combinations admitted by the original model are represented in the log, and not all the traces have the same frequency. Therefore,

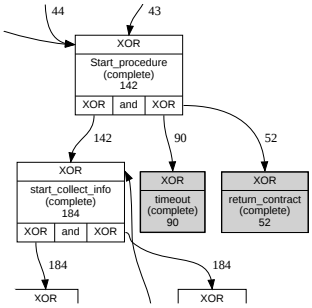
ProDiGen tries to better fit the most frequent behavior of the log, overfitting the data.



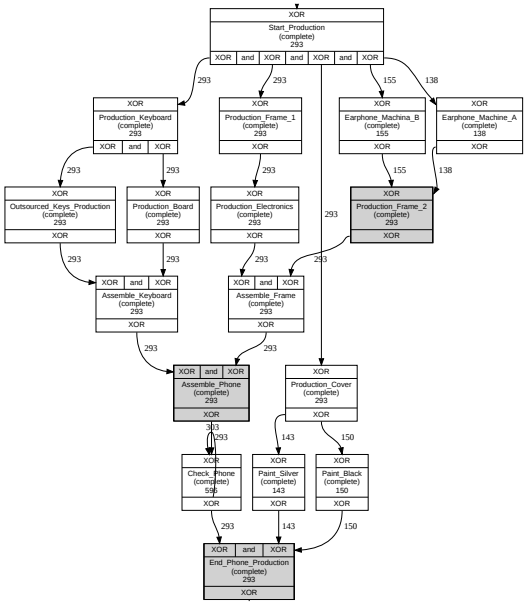
(a) *Detail of the mined model for the log g4.The tasks highlighted in grey are involved in a unbalanced AND-join/split point, making very difficult to correctly mine their inputs and outputs.*



(b) *Detail of the mined model for the log g24. In the original model, the missing relation is never used, therefore it is impossible to mine it.*



(c) *Detail of the mined model for the log g8. Both tasks hihglighted in grey are involved in an unbalanced AND-join/split.*



(d) *Heuristic net of the mined model for the log g25. The mined model incorrectly finds the AND-join point at the task "End_phone_product". This leads to other incorrect relations trying to better fit the log.*

Figure 6: Heuristics nets of the mined models for the unbalanced logs: *g4*, *g8*, *g24* and *g25*.

- The mined model for log *g*8 (Fig. 6c) has a behavioral precision and recall equal to 1. However, the model is not complete because it cannot tackle the output dependencies of the tasks *timeout* and *return-contract*, considering them as final tasks. This results in a incomplete mined model, because all the traces involving these two tasks will have an extra token at the end of the parsing.
- The results for log *g*24 (Fig. 6b) shows that the mined model is almost equal to the original one, except in only one relation between two tasks (tasks in grey, Fig. 6b).
- The mined model for log *g*4 (Fig. 6a) has a behavioral precision and recall of 1, i.e., the mined model allows the

same behavior as the original one with respect to the information contained in the log. The difference between the mined and original models is that the mined model cannot find —as with log $g8$— the output dependencies of the task *return_replacement_car*, therefore it considers that task as final, generating an extra final token every time the model parses a trace involving this task.

- For log $g25$ (Fig. 6d) the behavioral recall and precision are closer to 1. This means that, even when the model is not as precise as the original, it does not allow for too much extra behavior than the original one. The mined model cannot mine the original AND-join point.

Analyzing the results of the other algorithms, HM focuses its search on the main behavior of the log —finding solutions with high levels of simplicity. Hence, it cannot find the original model on those logs that came from models with many interleaving situations, as it tries to better fit the most frequent behavior recorded in the log —as the logs are unbalanced, *not all the possible relations* have the same frequency. On the other hand, ILP tends to retrieve complete models, but they usually are very general and, therefore, very different from the original model. Additionally, although ILP retrieves complete models when possible, it cannot tackle invisible tasks, giving poor results with the models containing this kind of constructs. With respect to Genetic Miner, based on its fitness definition —always benefits the individuals that portrait the most frequent behavior in the log—, it has problems to obtain complete and precise models when dealing with logs with many interleaving situations. This results in solutions with poorly precision values and very complex —with many silent tasks. Finally, the α^{++}-algorithm gets the worse results since it cannot deal with logs with many interleaving situations, giving as a result very poor values of completeness and precision for almost all the logs. Comparing the results of the five algorithms: ProDiGen correctly mines, i.e, finds the original model, the 81% (17 out of 21) of the cases. For the other algorithms, the percentage of correctly mined models was: GM in the 33% (7 out of 21), HM in the 28% (6 out of 21), and both α^{++} and ILP in the 5% (1 out of 21). Tables 7b and 7c show the results for the non-parametric tests for the unbalanced logs. Again, the p-value of the Friedman test is really low, indicating that there are significant differences among the algorithms with a high level of confidence. Holm's test also rejects the null hypothesis between ProDiGen and each of the algorithms used in the comparison. Therefore, we can conclude that ProDiGen also outperforms all the algorithms with unbalanced logs, and the difference is statistically significant. We also did the test with only the model metrics and only the log metrics —with no differences.

	ProDiGen			GM[7]	HM	α^{++}	ILP
	Initial Sol.	Best Sol.	Total	Total	Total	Total	Total
Average Time	2.5s	4.7m	32m	65m	606ms	128ms	33.8s

Table 8: Average runtimes of the algorithms on the 21 unbalanced logs.

[7]Genetic Miner is an iterative algorithm and, therefore, it would be possible to measure the runtimes of the first and best solutions and the total

Table 8 shows the average runtimes of all the algorithms over the logs of Sec. 5.1.2. As can be seen, Heuristics Miner, α^{++} and ILP have on average very fast runtimes, being the quality of the results of Heuristics Miner the highest of the three. On the other hand, ProDiGen improves the execution time of Genetic Miner. The comparison between the runtimes of ProDiGen and Heuristics Miner requires to take into account that ProDiGen is an iterative algorithm —it obtains thousands of solutions per execution—, while HM only gets one solution per execution. The *initial solution runtime* (Tab. 8) shows the time that ProDiGen needs, on average, to pre-process the log and retrieve de best solution of the initial population. The quality of that solution is at least as good as the solution of Heuristics Miner —as ProDiGen uses the Heuristics Miner solution as part of the initial population. Both the runtimes of the initial solution and HM are fast —as mining algorithms do not have real time requirements. Moreover, the extra-time inverted by ProDiGen in finding the *best solution* (Tab. 8) is worth, as ProDiGen improves the solutions of Heuristics Miner in 58 out of 111 logs —15 out of 21 unbalanced logs— in a fast way for process mining requirements.

6. Conclusions

We have presented ProDiGen, a genetic algorithm for process mining that can tackle all the different constructs at once, and obtains models that are complete, precise, and simple. ProDiGen uses a new hierarchical fitness function that includes new definitions for precision and simplicity. Moreover, the proposal uses genetic operators that focus the search on specific parts of the model: (i) the crossover operator selects the crossover point based on the errors of the mined model; and (ii) the mutation operator is guided by the causal dependencies of the log. ProDiGen has been validated with 111 different logs with all kind of workflow patterns, noise, and unbalanced logs. Also, we have compared ProDiGen with 4 of the state of the art process discovery algorithms using non-parametric statistical tests. Results conclude that using a hierarchical fitness based on completeness, precision and simplicity shows a great performance when retrieving the original model. Moreover, ProDiGen outperforms the other process mining algorithms, as it is able to retrieve the original model in the 84% of the tested logs.

Acknowledgment

This work was supported by the Spanish Ministry of Economy and Competitiveness under the project TIN2011-22935 and by the European Regional Development Fund (ERDF/FEDER) under the project CN2012/151 of the Galician Ministry of Education.

NOTICE: this is the authors version of a work that was accepted for publication in Information Sciences. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was

runtime. We only show the total runtime as the implementation of Genetic Miner (ProM 6.3) does not measure other runtimes. Nevertheless, the runtime for the initial solution is very similar to that of ProDiGen, although the quality of that solution for Genetic Miner is usually very low.

submitted for publication. A definitive version was subsequently published in Information Science, 100:315333, 2015, doi:10.1016/j.ins.2014.09.057.

[1] Agrawal, R., Gunopulos, D., , Leymann, F., 1998. Mining process models from workflow logs. Springer.

[2] Badouel, E., Darondeau, P., 1998. Theory of regions. In: Lectures on Petri Nets I: Basic Models. Springer, pp. 529–586.

[3] Bergenthum, R., Desel, J., Lorenz, R., Mauser, S., 2007. Process mining based on regions of languages. In: Business Process Management. Springer, pp. 375–383.

[4] Buijs, J., van Dongen, B., van der Aalst, W., 2012. On the role of fitness, precision, generalization and simplicity in process discovery. In: On the Move to Meaningful Internet Systems: OTM 2012. Springer, pp. 305–322.

[5] Burattin, A., Sperduti, A., 2010. Heuristics miner for time intervals. In: ESANN.

[6] Carmona, J., Cortadella, J., Kishinevsky, M., 2010. New region-based algorithms for deriving bounded petri nets. IEEE Transactions on Computers 59 (3), 371–384.

[7] Cook, J. E., Wolf, A. L., 1998. Discovering models of software processes from event-based data. ACM Transactions on Software Engineering and Methodology (TOSEM) 7 (3), 215–249.

[8] de Medeiros, A., 2006. Genetic process mining. Ph.D. thesis, Technische Universiteit Eindhoven.

[9] de Medeiros, A., van Dongen, B., van der Aalst, W., Weijters, A., 2004. Process mining: Extending the α-algorithm to mine short loops. Eindhoven University of Technology, Eindhoven 19.

[10] Dumas, M., ter Hofstede, A., van der Aalst, W., 2005. Process-aware information systems: bridging people and software through process technology. Wiley-Interscience.

[11] Ehrenfeucht, A., Rozenberg, G., 1990. Partial (set) 2-structures. Acta Informatica 27 (4), 343–368.

[12] Friedman, M., 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association 32 (200), 675–701.

[13] Goedertier, S., Martens, D., Vanthienen, J., Baesens, B., 2009. Robust process discovery with artificial negative events. The Journal of Machine Learning Research 10, 1305–1340.

[14] Greco, G., Guzzo, A., Ponieri, L., Sacca, D., 2006. Discovering expressive process models by clustering log traces. IEEE Transactions on Knowledge and Data Engineering 18 (8), 1010–1027.

[15] Greco, G., Guzzo, A., Pontieri, L., Sacca, D., 2004. Mining expressive process models by clustering workflow traces. In: Advances in Knowledge Discovery and Data Mining. Springer, pp. 52–62.

[16] Günther, C., van der Aalst, W., 2007. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: Business Process Management. Springer, pp. 328–343.

[17] Herbst, J., Karagiannis, D., 2004. Workflow mining with inwolve. Computers in Industry 53 (3), 245–264.

[18] Holm, S., 1979. A simple sequentially rejective multiple test procedure. Scandinavian journal of statistics, 65–70.

[19] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE transactions on Evolutionary Computation 6 (2) (2002) 182–197.

[20] Li, J., Liu, D., Yang, B., 2007. Process mining: Extending α-algorithm to mine duplicate tasks in process logs. In: Advances in Web and Network Technologies, and Information Management. Springer, pp. 396–407.

[21] Lorenz, R., Mauser, S., Juhás, G., 2007. How to synthesize nets from languages: a survey. In: Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come. IEEE Press, pp. 637–647.

[22] Maruster, L., 2003. A machine learning approach to understand business processes. Ph.D. thesis, Technische Universiteit Eindhoven.

[23] Murata, T., 1989. Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77 (4), 541–580.

[24] Rozinat, A., de Medeiros, A., Günther, C., Weijters, A., van der Aalst, W., 2007. Towards an evaluation framework for process mining algorithms. Beta, Research School for Operations Management and Logistics.

[25] Rozinat, A., van der Aalst, W., 2006. Decision mining in prom. In: Business Process Management. Springer, pp. 420–425.

[26] Rozinat, A., van der Aalst, W. M., 2008. Conformance checking of processes based on monitoring real behavior. Information Systems 33 (1),

64–95.

[27] Sánchez-González, L., Garca, F., Mendling, J., Ruiz, F., M.Piattini, 2010. Prediction of business process model quality based on structural metrics. In: Conceptual Modeling ER 2010. Vol. 6412 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 458–463.

[28] Solé, M., Carmona, J., 2010. Process mining from a basis of state regions. In: Applications and Theory of Petri Nets. Springer, pp. 226–245.

[29] van der Aalst, W., 2011. Process mining: discovery, conformance and enhancement of business processes. Springer.

[30] van der Aalst, W., Adriansyah, A., van Dongen, B., 2012. Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2 (2), 182–192.

[31] van der Aalst, W., Rubin, V., van Dongen, B., Kindler, E., Günther, C., 2006. Process mining: A two-step approach using transition systems and regions. BPM Center Report BPM-06-30, BPMcenter. org.

[32] van der Aalst, W., Ter Hofstede, A. H., Kiepuszewski, B., Barros, A. P., 2003. Workflow patterns. Distributed and parallel databases 14 (1), 5–51.

[33] van der Aalst, W., van Dongen, B., 2002. Discovering workflow performance models from timed logs. In: Engineering and Deployment of Cooperative Information Systems. Springer, pp. 45–63.

[34] van der Aalst, W., Weijters, A., Maruster, L., 2004. Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16 (9), 1128–1142.

[35] van der Werf, J., van Dongen, B., Hurkens, C., Serebrenik, A., 2008. Process discovery using integer linear programming. In: Applications and Theory of Petri Nets. Springer, pp. 368–387.

[36] van Dongen, B., de Medeiros, A., Verbeek, H., Weijters, A., van der Aalst, W., 2005. The ProM framework: A new era in process mining tool support. In: Applications and Theory of Petri Nets 2005. Springer, pp. 444–454.

[37] van Dongen, B., de Medeiros, A., Wen, L., 2009. Process mining: Overview and outlook of petrinet discovery algorithms. In: Jensen, K., van der Aalst, W. (Eds.), Transactions on Petri Nets and Other Models of Concurrency II. Vol. 5460 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 225–242.

[38] vanden Broucke, S., Weerdt, J. D., Vanthienen, J., Baesens, B., 2013. A comprehensive benchmarking framework (CoBeFra) for conformance analysis between procedural process models and event logs in prom. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM). IEEE, pp. 254–261.

[39] Weijters, A., van der Aalst, W., 2003. Rediscovering workflow models from event-based data using little thumb. Integrated Computer-Aided Engineering 10 (2), 151–162.

[40] Weijters, A., van der Aalst, W., de Medeiros, A., 2006. Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven 166.

[41] Wen, L., van der Aalst, W., Wang, J., Sun, J., 2007. Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery 15 (2), 145–180.

[42] Wen, L., Wang, J., Sun, J., 2007. Mining invisible tasks from event logs. In: Advances in Data and Web Management. Springer, pp. 358–365.