

X-Processes: Discovering More Accurate Business Process Models with a Genetic Algorithms Method

Marcelo Fantinato
School of Arts, Sciences
and Humanities
University of São Paulo
São Paulo, Brazil
m.fantinato@usp.br

Sarajane Marques Peres
School of Arts, Sciences
and Humanities
University of São Paulo
São Paulo, Brazil
sarajane@usp.br

Hajo A. Reijers
Department of Information
and Computing Sciences
Utrecht University
Utrecht, The Netherlands
h.a.reijers@uu.nl

Abstract—Although process model discovery has been extensively investigated over the past two decades, existing discovery methods are still not considered fully satisfactory. One problem is the difficulty of discovering accurate process models, achievable with both high recall (or fitness) and high precision, particularly for real-world event logs. This paper introduces a process discovery method, namely X-Processes, based on genetic algorithms, which aims to optimize accuracy through the F-Score calculated between recall and precision. Although genetic algorithms have been used to discover process models, such methods also have limitations as do other non-genetic algorithms-based methods. Experimental results for 12 real-world event logs show the accuracy of the process models discovered by X-Processes is higher than those of six other state-of-the-art discovery methods, including one also based on genetic algorithms. Besides accuracy, X-Processes delivers sound process models. Although its execution time is longer than the other compared discovery methods, X-Processes emerges as a solution when the need for a highly accurate process model outweighs the hunger for agility.

Index Terms—process mining, automated process discovery, genetic algorithms, fitness, precision, accuracy

I. INTRODUCTION

Discovering a model that reflects the actual execution of a business process is one of the most challenging process mining tasks [1]. Process model discovery is based on the analysis of event logs, where the occurrences of activities performed by process instances in a time frame are recorded. Since the raise of the classical α -algorithm [2], discovery methods have been extensively investigated [3]–[10]. While they rely on particular heuristics, they all pursue a same goal: discovering a high-quality process model from real-world logs. Their main challenge is to discover highly accurate models, i.e., with both high recall (or fitness) and high precision [10], seeing that a model with high accuracy is able to represent all behavior, and only that behavior, present in the log. In fact, a well-fitted model is useless if too imprecise. Given the accuracy relevance, Augusto et al. [7] use it as the key criterion to assess the quality of discovered models, based on the F-Score between recall and precision. In addition, the discovered models are expected to be sound, i.e., behavioral error free [11].

This research was funded by the São Paulo Research Foundation (Fapesp), Brazil, under grant numbers 2017/26491-1 and 2020/05248-4.

This paper introduces a novel process discovery method, namely X-Processes¹. X-processes is a genetic algorithms-based method designed to optimize *accuracy* through the F-Score calculated between *recall* and *precision*. We do not just use accuracy to assess the quality of the discovered models as Augusto et al. [7], but also to guide the process discovery way. Besides being accurate, X-Processes delivers sound process models. X-Processes was empirically tested on 12 real-world logs. The results show the X-Processes' superiority for accuracy compared to the six discovery methods used as a baseline. X-Processes does not handle generalization and simplicity optimization and still has room for execution time improvement. Directions for addressing both limitations are briefly outlined at the paper conclusion.

The rest of this paper is structured as follows. Section II presents basic concepts, an overview of process model quality criteria and automated process discovery methods. Section III details the X-Processes method. Section III reports the empirical evaluation of X-Processes. Finally, Section V concludes the paper, including directions for future work.

II. BACKGROUND AND RELATED WORK

A. Overview on Genetic Algorithms

In a traditional genetic algorithm, a fixed-length bit vector is used as the chromosome representation of individuals. Each position represents an individual's feature, whereas the position's value represents how this feature is expressed in the individual. The main genetic operators employed in genetic algorithms are [12], [13]: (i) crossover, when two individuals of the current population are selected for reproduction, and new individuals are generated by combining the bit segments of their vectors and (ii) mutation, when the value of a single bit of the selected vector is altered so that it inhibits or expresses the feature of such a bit and thus generates a new individual.

To begin evolution, genetic algorithms often generate an initial population of individuals at random. Evolution refers to selecting individuals for reproduction, producing variations of them, assessing the quality of the offsprings and choosing

¹The term *X-Processes* is an allusion to the *X-Men* comic books to represent process models resulting from the evolutionary process.

the ones that should compose the new generation. This cycle is repeated for a predetermined number of generations or until an individual with an expected fitness is got. During the evolution cycle, crossovers and mutations are executed to increase population diversity. The general objective of a genetic algorithm is to get better populations throughout each generation according to fitness [12], [13].

B. Process Mining Basic Concepts

Process mining relies on the concepts of events, cases, traces, and logs [1]. An *event* is the occurrence of a process activity at a given time. A *case* refers to a process instance and comprises events such that each event relates exactly to a case. A *trace* is a mandatory attribute of a case and refers to a finite sequence of events such that each event appears only once. An *event log* (or simply *log*) is a set of cases such that each event appears at most once in the entire log.

Discovered process models can be represented with different notations. In this paper, we use *workflow nets* (WF-nets), a particular class of Petri nets, which have become one of the standard ways to model and analyze process models [11].

As with other discovery methods, we use the concept of *direct follow graph* (DFG), which relies on the notion of the *directly-follows relation* (DF-relation) [1]. A DF-relation from event *A* to event *B* holds if *A* directly precedes *B* in at least one trace in the log. A DFG is a graph with nodes that correspond to activities (derived from the events in the log) and directed edges that correspond to the DF-relations.

We also use the concept of *causal matrix*. A causal matrix of a process model describes the causal dependencies between the activities of the process [8]. The causal matrix shows which activities enable the execution of which other activities, considering the matching of input and output condition functions, which are represented by input and output gateways.

C. Process Model Quality Criteria

Discovered process models are often assessed through four competing quality criteria [1]. *Recall* (or *fitness*²) assesses whether a process model can reproduce the behavior contained in the log. A process model is fully complete whether all the traces in the log can be replayed by the model from beginning to end. *Precision* assesses whether a process model can generate only the behavior present in the observed process. A process model is fully precise whether any trace produced by the model is present in the observed process. The available log is commonly regarded as a representative sample of the observed process. *Generalization* assesses whether a process model can generate behavior present in the observed process, but not contained in the log. A process model is fully generic whether all the behavior of the observed process is represented in the process model. *Simplicity* assesses whether a process model is easy to understand. A process model should describe the behavior in the log with as few elements as possible.

²Although in the context of process mining the term *fitness* is used more often than *recall*, we avoid it herein to differentiate this quality criterion from the *fitness function* in the context of genetic algorithms.

A derived quality criterion is *accuracy*, which considers recall and precision together [7]. Accuracy can be calculated as the harmonic mean between recall and precision, usually called *F-Score*. This criterion measures in a combined way how much a process model represents the log.

Specific measures have been proposed for each quality criteria. In this paper, we adopted the state-of-the-art measures implemented in the PM4PY library³. The four measures implemented in PM4PY result in a number from 0 to 1. Thus, for recall and precision, the measures proposed by Adriansyah et al. [14], [15] are respectively used. These two measures are based on trace alignment principles and measure the degree to which each trace in the log can be aligned with a corresponding trace produced by the process model. For generalization, the measure described by Buijs, Van Dongen and Van der Aalst [10] is used. This measure is based on the frequency of times the model elements are visited while replaying the log. The higher the frequency, the greater the model generalization. For simplicity, the inverse arc degree of a Petri net, as described by Rojas [16], is used. This measure is based on the Petri net's place and transition degrees, i.e., the sum of the number of input and output arcs of places and transitions. The smaller the Petri net's average degree, the greater its simplicity.

An additional way to evaluate process model quality is through *soundness* [11]. Soundness is related to behavioral correctness, as it guarantees the absence of livelocks, deadlocks, and other anomalies that can be detected without domain knowledge. A WF-net is sound if and only if the following three requirements are satisfied: *option to complete* – for each case, it is always possible to reach the end place, *proper completion* – if the end place has a token, all other places are empty for a given case, and *no dead transitions* – it should be possible to execute an arbitrary activity by following the appropriate route through the WF-net.

D. Process Discovery Methods

The α -algorithm [2] is a DFG-based method considered one of the forerunners of discovery methods. Although quite simple, the α -algorithm is not considered an effective method due to issues with noise, infrequent and incomplete behavior, and complex routing constructs. Heuristics Miner [3] (HM) considers noise and frequency when building a process model, preventing infrequent paths from being incorporated into the model. Although HM commonly exhibits fairly good accuracy in the presence of noise, it produces large and often unsound models for real-world logs. Fodina [4] is a variant of HM that avoids certain types of deadlocks, but still produces large and often unsound models. Inductive miner [5] ensures soundness through a divide-and-conquer approach, as it relies on inherently structured process trees [1]. While IM exhibits high recall, it loses precision when the underlying behavior of the log is unstructured. Structured Miner over Heuristics Miner (S-HM) [6] applies techniques to maximally structure, in

³<http://pm4py.fit.fraunhofer.de>, version 2.2.4.

addition to heuristics to simplify a model discovered through HM, which is often accurate but unstructured. However, S-HM also often fails to produce sound model for real-world logs. Split Miner (SM) [7] is the latest of the discovery methods listed herein. SM produces simple, sound models with high accuracy and is therefore considered herein the best state-of-the-art discovery method. SM combines a novel approach to filter the DFG induced by a log, with an approach to identify combinations of split gateways that capture the concurrency, conflict and causal relations between neighbors in the DFG.

As for genetic algorithms, an individual represents a process model to be discovered. Thus, a suitable chromosome representation should be used to properly describe a process model structure. The first proposed genetic algorithms-based discovery method used a symbolic causal matrix as chromosome, which does not guarantee soundness [8]. This method was later extended [9] to improve processing time by introducing an island model-based distributed approach, in addition to incremental sampling of the log. Evolutionary Tree Miner (ETM) [17] is a more recent method that uses process trees as chromosome, which inherently guarantees soundness. However, ETM presents polluted models that contradict good modeling practices, e.g., showing the same activity multiple times in different parts of the model. In terms of recall and precision, ETM showed better results [10] than the first proposed method [8]. There are still other methods proposed, but without comparable recall and precision results.

III. THE X-PROCESSES METHOD

The X-Processes method implements the standard flow for genetic algorithms (cf. Fig. 1). What primarily characterizes it as a process mining domain solution is the initial population creation and the population assessment both based on log. The algorithm was implemented using parallelism, following an island-based model [18] for computational efficiency purposes.

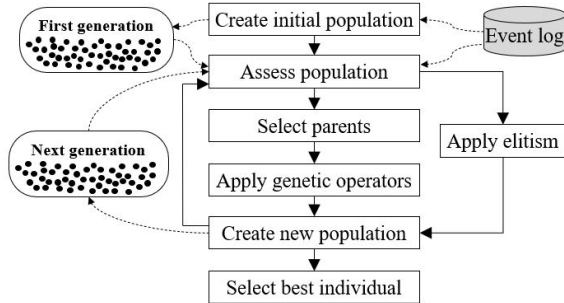


Fig. 1. X-Processes' algorithm flow.

This section details the key steps and features of the implemented algorithm and island-based model, starting with the chromosome representation that supports the entire method.

A. Chromosome Representation

The chromosome is represented by a binary causal matrix. Consider in Fig. 2 the example of a process model in WF-net. The causal matrix in Fig. 3 shows the chromosome representation of the model in Fig. 2.

The chromosome comprises a binary matrix $M_{(n+3) \times (n+3)}$, where n is the number of distinct activities in the log. As a general rule, cells show whether there is a DF-relation from an activity represented by a row (*Input*) to an activity represented by a column (*Output*): 1 in the cell shows there is a DF-relation, and 0 shows none DF-relation. For example, there is a DF-relation between activity *D* (as *Input*) and activity *E* (as *Output*), cf. Fig. 2 and Fig. 3. On the other hand, also as an example, there is no DF-relation between activity *D* (as *Input*) and activity *G* (as *Output*).

In addition to the n process activities in the log, the chromosome represents two synthetic activities – *Begin* and *End*. These two extra activities are needed to ensure the discovered process model has only one beginning activity and only one end activity, as required by WF-nets. The first column of the matrix is not used, as the synthetic activity *Begin* never directly follows any other activity. Likewise, the penultimate row of the matrix is not used, since any other activity never directly follows the synthetic activity *End*. Although not used, they are kept in the matrix to facilitate implementation by allowing to match activity-related indexes (i.e., a same index value for column and row for each activity).

The last column and the last row of the matrix show the gateway type, where appropriate: 0 represents *XOR* and 1 represents *AND*. The gateway type is required when there are at least two outputs for the same input, or at least two inputs for the same output. For example, both *B* and *C* follows directly *A*; here, 0 in the last column of row *A* means *XOR*, i.e., *B* and *C* follows directly *A* with a *XOR* gateway. Likewise, both *F* and *G* follows directly *E*, but with an *AND* gateway. For input gateways, the usage is similar. For example, *D* follows directly *B*, *C* and *H*; here, 0 in the last row of column *D* means *XOR*, i.e., *D* follows directly *B*, *C* and *H* via a *XOR* gateway. Likewise, *H* follows directly *F* and *G*, but via an *AND* gateway. When there is a single DF-relation for a given row (e.g., for row *B*, since only *D* directly follows *B*), the bit relative to the output gateway (i.e., the last cell of such a row) does not matter and is set to 0 by convention. The same applies when there is a single DF-relation for a given column (e.g., for column *F*, since *F* directly follows only *E*), the bit relative to the input gateway (i.e., the last cell of such a column) does not matter and is also set to 0.

Loops can also be represented on the chromosome. For example, activity *J* has a self-loop, represented by 1 in the cell at row *J*, column *J* (i.e., *J* as both input and output of the DF-relation). A longer loop is represented by *D* directly following *H*, which creates a repetition for activities from *D* to *H*. For both situations, although a silent transition is necessary in the graphical representation in a WF-net, the corresponding representation in the causal matrix is straightforward.

This chromosome representation has advantages over the other two main representations in the literature. Compared to a *process tree* [19], a causal matrix ensures each activity appears only once in the discovered process model, delivering leaner models that do not contradict good process modeling practice. Compared to a *symbolic causal matrix* [8], a binary

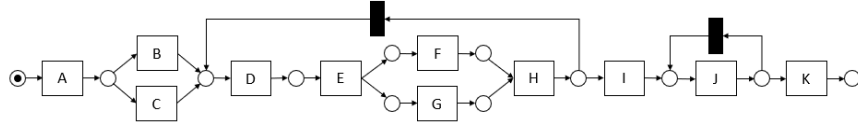


Fig. 2. An illustrative example of a process model represented in a WF-net.

		Output											
Input	Begin	A	B	C	D	E	F	G	H	I	J	K	End
	Begin	1	0	0	0	0	0	0	0	0	0	0	0
	A	0	1	1	0	0	0	0	0	0	0	0	0
	B	0	0	0	1	0	0	0	0	0	0	0	0
	C	0	0	0	1	0	0	0	0	0	0	0	0
	D	0	0	0	0	1	0	0	0	0	0	0	0
	E	0	0	0	0	0	1	1	0	0	0	0	1
	F	0	0	0	0	0	0	0	1	0	0	0	0
	G	0	0	0	0	0	0	0	1	0	0	0	0
	H	0	0	0	1	0	0	0	0	1	0	0	0
	I	0	0	0	0	0	0	0	0	0	1	0	0
	J	0	0	0	0	0	0	0	0	0	1	1	0
	K	0	0	0	0	0	0	0	0	0	0	0	1
	End	0	0	0	0	0	0	0	0	0	0	0	0
Gateway		0	0	0	0	0	0	0	1	0	0	0	0

Fig. 3. Example of chromosome representation by causal matrix, for the process model illustrated in Fig. 2.

causal matrix increase efficiency, which is recognized as the key challenge for genetic algorithms-based solutions.

B. Direct-Follow Graph

A DFG of the log is created as a chromosome and used by the genetic algorithm to decrease the search space by avoiding generating poor quality individuals. Although a DFG is known not to deliver good quality (cf. the quality criteria in Section II), it works as a sort of mask for the individuals created or manipulated in the algorithm's operations, as explained later.

As an example, see the illustrative log in Fig. 4. Fig. 5 shows the chromosome representation for the DFG of the log in Fig. 4. All DF-relations in the log are represented on the chromosome with 1, plus the extra DF-relations from *Begin* as well as the extra DF-relations to *End*. As XOR or AND gateways do not apply to DFG, all cells in the last row and last column of the chromosome are set to 0.

[<A, B, E, F, B>,
 <A, E, D, F>,
 <B, D, G, G, H, G>,
 <A, B, E, G, H, G>,
 <B, D, F, B, E, F, F>,
 <B, C, D, H>]

Fig. 4. Illustrative example of log.

DFG											
Beg		A	B	C	D	E	F	G	H	End	Gat
Beg	A	1	1	0	0	0	0	0	0	0	0
	B	0	1	0	0	1	0	0	0	0	0
	C	0	0	1	1	1	0	0	0	1	0
	D	0	0	0	1	0	0	0	0	0	0
	E	0	0	0	0	0	1	1	1	0	0
	F	0	0	0	1	0	1	1	0	0	0
	G	0	1	0	0	0	1	0	0	1	0
	H	0	0	0	0	0	0	1	1	1	0
	End	0	0	0	0	0	0	1	0	1	0
	Gat	0	0	0	0	0	0	0	0	0	0

Fig. 5. Chromosome for the DFG of the log in Fig. 4. Red cells represent the DFG mask.

A DFG acts as a constraint to possible solutions in the search for good quality process models for the log. Thus, any solution must be within the red highlighted part of the DFG (i.e., with 1). In the search for the best solution, a given chromosome can have 0 where the DFG has 1, which would mean the nonexistence in such a chromosome of a particular DF-relation present in the DFG. However, a chromosome can never have 1 where the DFG has 0, which would mean the existence in such a chromosome of a particular DF-relation not present in the DFG (nor in the log).

C. Input Event Log

The proposed method does not take into account the trace frequency. Thus, the input log is pre-processed to keep only one sample of each trace. Likewise, the frequency of activities in a self-loop is disregarded; hence, only two identical activities in sequence are kept in the log. These pre-processing actions are carried out to decrease the log size and, in this way, to decrease the method execution time.

D. Initial Population

Chromosome initialization is based on heuristics that include randomness to allow for diversity. These heuristics increase the quality of the process model, in addition to helping to achieve soundness, although this is not guaranteed.

DF-relations are created, originating a certain number of paths from *Begin* to *End*, provided the following constraints are met: (1) all DF-relations must meet the DFG mask; (2) all DF-relations must be on a path from *Begin* to *End*; and (3) every activity must directly follow at least one other activity as well as be directly followed by at least one other activity.

For gateway types, the total number of tokens⁴ produced for all rows must be equal to the total number of tokens consumed for all columns. If they are different, the gateway types change

⁴The *token* notion used here is the same as that used in WF-nets.

randomly, one at a time, until the total numbers of tokens produced and consumed are equal.

In the end, each resulting individual representing a process model must be sound. To ensure that, each individual created is subjected to a soundness test⁵. If any unsound model is found, it is discarded and a new one is created.

E. Fitness Function

The fitness function assesses the quality of each individual in the population considering its *accuracy*, i.e., its *recall* and its *precision* combined, with respect to the log (cf. Section II). The fitness function is calculated through a harmonic mean (called F_score), according to Eq. 1.

$$F_score_{accuracy} = 2 * \frac{(recall * precision)}{(recall + precision)} \quad (1)$$

Recall and precision are calculated based on the alignment-based algorithms [14], [15]⁶. To perform the calculations, these alignment-based algorithms require sound process models as input, which is guaranteed by the X-Processes method.

F. Genetic Operators

1) *Crossover*: a specific crossover method for our chromosome representation was implemented, being called *activity-driven crossover*. Through the activity-driven crossover, two activities are fully swapped with each other for two parent chromosomes, so that all data in the row (as well as in the column) corresponding to that activity on both chromosomes are swapped with each other. As a result of this crossover operation, all DF-relations for a given activity are swapped between two chromosomes, cell by cell. A random number of up to $n - 1$ activity pairs are chosen to be swapped for each crossover run, where n is the number of activities in the log. For example, in Fig. 6, activities C , G and H are swapped between parent chromosomes 1 and 2, generating offspring chromosomes 1 and 2. All data of activity C on parent chromosome 1 (i.e., row and column in light orange) are swapped with the data of the corresponding activity C on parent chromosome 2 (i.e., row and column in dark orange). The same is done for activity G (i.e., for light and dark green rows and columns) as well as for activity H (i.e., for light and dark blue rows and columns). Note that crossover swaps never violate the DFG mask. Other crossover options implemented are: *classic single point*, *classic two points*, and *classic uniform* [12]. However, as test runs did not show encouraging results for their use, they were not applied in the end.

2) *Mutation*: two variations of the classic single mutation [12] are used – *DF-relation mutation* and *gateway type mutation*. In either case, mutation just means changing 0 to 1, or vice versa. For *DF-relation mutation*, bit-changing means

⁵For the soundness test, we use the WOFLAN [20] implementation provided in PM4PY. As WOFLAN requires WF-net as input, a procedure was developed to convert a process model represented by a chromosome into a WF-net, using PM4PY's Petri net management implementation.

⁶We use PM4PY implementations of both algorithms. As they require WF-net as input, the procedure to convert the chromosome into WF-net was used.

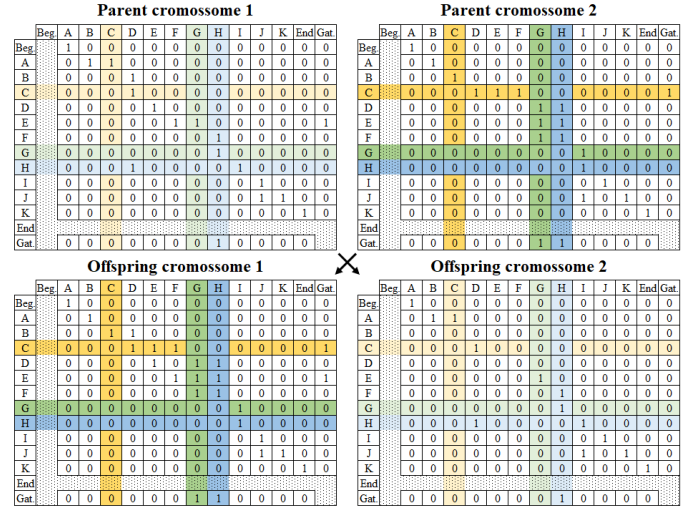


Fig. 6. Illustrative example of activity-driven crossover.

to create or break a DF-relation between two activities. For example, in Fig. 3, a mutation in the cell at row G , column C (changing 0 to 1) means to create a DF-relation between activities G and C , meaning C directly follows G . On the other hand, a mutation in the cell at row H , column D (changing 1 to 0) means to break the DF-relation between H and D . For mutations that create a new DF-relation, the DFG mask must be met. Thus, a mutation that would create a DF-relation not present in the DFG is discarded. A mutation that breaks DF-relations never violates the DFG mask. A random number of up to n activities is chosen to undergo the DF-relation mutation, where n is the number of activities in the log. For *gateway type mutation*, bit-changing means to change the gateway type (XOR to AND , or vice versa), either of an input or output gateway. As an example, also in Fig. 3, a mutation in the cell at row A , column *Gateway* (changing 0 to 1) means to change the output gateway type from XOR to AND . As another example, a mutation in the cell at row *Gateway*, column H (changing 1 to 0) means to change the input gateway type from AND to XOR . Gateway type mutation does not apply to rows with a single DF-relation as in such cases there is no output gateway for the corresponding bit to mutate (e.g., row B , as only D directly follows B , cf. Fig. 3). Likewise, gateway type mutation does not apply to columns with a single DF-relation as in such cases there is no input gateway (e.g., column F , since F directly follows only E). Again, gateway type mutations do not violate the DFG mask. Similar to the DF-relation mutation, a random number of up to n activities is chosen to undergo the gateway type mutation, where n is the number of activities in the log. Directed (guided) mutation [21] was also implemented. However, test runs did not show encouraging results for its use and hence this mutation option was not maintained.

3) *Unsound Individuals*: chromosomes resulting from crossover or mutation may become unsound due to an imbalance between the total numbers of produced and consumed

tokens (as in the initialization of individuals, cf. Section III-D). When that happens, to help bring these individuals back to soundness, the gateway types are changed randomly, one at a time, until the total number of tokens produced for all rows is equal to the total number of tokens consumed for all columns.

G. Evolution Flow

To guide the *selection of parents* for the application of crossover followed by mutation, the classic tournament strategy [12] is used. The classic roulette strategy [12] was also tested, but did not show promising results.

When *creating new population*, parents selected to participate in crossover and mutation, by default, do not pass on along with their offspring to the next generation of individuals to allow for population change. However, to enable *elitism*, a number of the best individuals of a current generation may have a chance to remain in the population and hence be moved on to the next generation of individuals.

After crossover and mutation, *soundness* is checked. If both offspring chromosomes do not correspond to sound process models, they are discarded and crossover and mutation are executed again until two sound models emerge.

As *stopping criteria*, the flow ends when it reaches a given: execution time, total number of generations, or consecutive number of generations with no fitness improvement for the best individual. The individual with the best fitness is *selected as the best individual*, i.e., the best discovered process model.

H. Island Model

The island model [18] allows the distributed execution of parallel copies of a process to improve the execution efficiency of an algorithm. For genetic algorithms, the island model allows subpopulations to evolve independently with results faster than all individuals evolving as a single population. Several small competing populations with occasional migrations display better search performance than a large population with the same size in total [18]. Thus, in the island model, the population is partitioned into isolated subpopulations that must evolve independently to maximize the same *fitness* function. Migration between subpopulations must occur periodically so that some individuals from each subpopulation are exchanged, considering a well-defined neighborhood structure.

In our solution, a population of p individuals is split into q subpopulations, which evolve independently on q islands, each island running on a processor core. Each subpopulation can have a different size of r individuals. To maximize the chances of exchanging genetic material between all pairs of islands during migrations, two types of topology are adopted:

- *Broadcast*: A controller mediates migrations. Every s generations, the $t\%$ best individuals on an island are sent to a controller to be sent later to other islands; while the $u\%$ worst individuals from that island are replaced by the best individuals from other islands (randomly chosen) previously sent to the controller.

- *Point-to-point*: every s generations, the best individual on an island is sent to another island (randomly chosen), replacing the worst individual of that island.

Each island can have different values for r , s , t , and u , allowing to explore different profile settings for each island.

IV. EVALUATION

We run X-Processes⁷ on a set of 12 publicly available logs. The results got by X-Processes were empirically compared to six process discovery methods using the process models discovered for the same logs cf. a previous study [7].

A. Event logs

A set of 12 real-world logs from the well-known collection provided by the 4TU Centre for Research Data were used in the evaluation of X-Processes⁸. Table I summarizes the data describing the logs used. Most of the logs used come from the annual Business Process Intelligence Challenge (BPIC). In addition to them, the Road Traffic Fines Management Process (RTFMP) and the SEPSIS Cases logs were used. These logs refer to different domains including healthcare, finance, and government. They are very heterogeneous in terms of number of traces, cases, events, distinct activities, and trace length.

TABLE I
DATA DESCRIBING THE LOGS USED [7]

Log name	Cases	Traces	Events	Distinct activities	Trace length		
					Min	Avg	Max
BPIC12	13,087	4,366	262,200	24 [†]	3	20	175
BPIC13_cp	1,487	183	6,660	4 [†]	1	4	35
BPIC13_inc	7,554	1,511	65,533	4 [†]	1	9	123
BPIC14 [§]	41,353	14,948	369,485	9	3	9	167
BPIC15_1 [§]	902	295	21,656	70	5	24	50
BPIC15_2 [§]	681	420	24,678	82	4	36	63
BPIC15_3 [§]	1,369	826	43,786	62	4	32	54
BPIC15_4 [§]	860	451	29,403	65	5	34	54
BPIC15_5 [§]	975	446	30,030	74	4	31	61
BPIC17 [§]	21,861	8,767	714,198	41	11	33	113
RTFMP	150,370	231	561,470	11	2	4	20
SEPSIS	1,050	846	15,214	16	3	14	185

[†]These three logs originally have a classifier *concept:name & lifecycle:transition*, but the experiments were run only with *concept:name* by both Augusto et al. [7] and hence by us. Augusto et al. reported the numbers of distinct activities considering the unused original binary classifier, while we report the numbers considering only the *concept:name* classifier.

[§]These seven logs had been pre-processed and filtered by Augusto et al. [7] to remove infrequent behavior. Accordingly, we used the same pre-filtered versions to ensure we would use exactly the same input data.

B. Experimental setup

X-Processes is compared to six state-of-the-art process discovery methods: SM, IM – infrequent (IM_F), ETM, ProM 6's HM (HM₆), ProM 6's S-HM (S-HM₆), and FO.

The parameter values used by X-Processes in this experiment are listed in Table II. The first part of Table II

⁷Python code available at <https://drive.google.com/open?id=1YVDxxVO-puTJLxLTN1pbNxDS5dfuDHddq>

⁸Available at <https://data.4tu.nl>

lists the values of the genetic operators and elitism-related parameters, adopted equally for all islands. Different values were empirically tested for these seven parameters; the values listed are those found to give the best results for the logs used in this experiment. Therefore, there was no systematic search for the optimal parameters for this set of logs, let alone for each log specifically. The second part of Table II lists the values of the island model-related parameters. A total of 100 individuals were used, split equally into 20 islands of five individuals each. Islands with different sizes were not used in this experiment, as the *BPIC12* log has characteristics that make its processing very slow and hence islands with a larger number of individuals would take a long time to change generation for this log. Thus, to allow for equal running conditions, this setting was used for all 12 logs, although other settings could favor the processing of other logs. Migration time was set to the minimum of one generation to all islands for the same reason, i.e., to allow migrations to the *BPIC12* log to occur even with just a few generations. For each island, values for *Percentage of best individuals for migration* and *Percentage of worst individuals for migration* were randomly set considering 0.4, 0.5, and 0.6. The third part of Table II lists the values of the stopping criteria-related parameters, i.e., maximum execution time, maximum number of generations, and convergence criterion, whichever comes first. The convergence stopping criterion was defined as the maximum consecutive number of generations with no fitness improvement for the best individual.

TABLE II
VALUES OF THE PARAMETERS USED IN THE EXPERIMENT

Parameter	Value
Genetic operators and elitism-related parameters	
Crossover probability	0.7
Maximum percentage of activity pairs for crossover	0.7
DF-relation mutation probability	0.7
Maximum percentage of activities for DF-relation mutation	0.5
Gateway type mutation probability	0.7
Maximum percentage of activities for gateway type mutation	0.3
Elitism percentage	0.3
Model island-related parameters	
Number of islands (or subpopulations)	20
Subpopulation size (in individuals)	5
Migration time (in generations)	1
Percentage of best individuals for migration	0.4; 0.5; 0.6
Percentage of worst individuals for migration	0.4; 0.5; 0.6
Stopping criteria-related parameters	
Maximum execution time (in hours)	24
Maximum number of generations	5,000
Convergence criterion (in generations)	5

The quality of the discovered process models was evaluated using the four quality criteria discussed in Section II, namely, *recall*, *precision*, *generalization*, and *simplicity*. These quality criteria were calculated using methods implemented by PM4PY. As alignment-based recall [14] and precision [15] are used, discovered unsound models were discarded. *Accuracy* was calculated based on recall and precision through *F-Score*.

We did not use exactly all the same quality criteria as Augusto et al. [7], as we rely on PM4PY implementations. Instead, we just used the models discovered through their study, available in a repository, and then recalculated the quality criteria using PM4PY⁹, ensuring the evaluation of all models discovered under the same conditions. Execution time is also reported.

Augusto et al. [7] adopted the following procedure to parameterize the discovery methods: SM was exhaustively optimized with the best parameters for the 12 logs, while the default parameters were used for the other five discovery methods (IM_F, ETM, HM₆, S-HM₆, and FO).

Five X-Processes runs were made for each log. After all, a probabilistic approach can yield different results on each run¹⁰. For each log, we report the best result, in terms of F-Score-driven accuracy, as well as the mean and standard deviation of the five results. The experiment was performed on a 40-core 3.1GHz Intel Xeon Gold 6242R with 128GB of RAM.

C. Results

Table III shows the accuracy results from the evaluation. The best score for each criterion on a given log is highlighted in bold. For discovered unsound process models, an “–” is shown as quality metrics were not calculated for such cases. Although we used the same measures as Augusto et al. [7] to calculate recall and precision, only the recall results are equal for SM, IM_F, HM₆, S-HM₆, and FO¹¹, for our experiment and theirs. The precision results differ (for greater and lesser) likely because different measure implementations are used for this criteria, which may rely on parameters and replay algorithms. Accordingly, the calculated F-Scores (for accuracy) also differ. Although X-Processes has the best F-scores, it does not necessarily have the best recall and precision scores. In fact, to achieve the best F-Score, X-Processes seeks a balance between recall and precision through its fitness function. Thus, by giving up pursuing the best individual scores for both recall and precision, X-Processes realizes a better F-Score. Although seeking better accuracy based on a balance between recall and precision is also the basis of SM, it showed lower F-Score results for the same logs. Moreover, except for the *BPIC12* log, X-Processes achieved F-Score greater than 0.9 for all other logs, which is not the case for SM.

Table IV shows the generalization and simplicity results. Like before, the best scores are highlighted in bold; also, an “–” is shown for discovered unsound process models. X-Processes did not get the best scores for generalization and simplicity for any log. In fact, these two quality criteria were not optimized by the genetic algorithm’s fitness function so as not to hinder the F-Score optimization, taken as priority. However, an adapted fitness function can also include gener-

⁹Exclusively for ETM only, we had to perform the process discovery again as those discovered models available in the consulted repository could not be processed by PM4PY due to some unknown error.

¹⁰Produced results available at https://drive.google.com/open?id=1VzBvX2CAVDq36sDbi_IOSaUH_HwoGTHI

¹¹Except for ETM since we had to rediscover the process models, and getting the same results would not be expected for a probabilistic approach.

TABLE III
ACCURACY-RELATED EXPERIMENTAL RESULTS

Event log	X-Processes			SM			IM _F			ETM			HM ₆			S-HM ₆			FO		
	F-Sc.	Rec.	Prec.	F-Sc.	Rec.	Prec.	F-Sc.	Rec.	Prec.	F-Sc.	Rec.	Prec.	F-Sc.	Rec.	Prec.	F-Sc.	Rec.	Prec.	F-Sc.	Rec.	Prec.
BPIC12	0.857	0.781	0.949	0.660	0.963	0.502	0.277	0.977	0.162	0.474	0.332	0.827	—	—	—	—	—	—	—	—	—
BPIC13_cp	0.981	0.988	0.973	0.911	0.988	0.845	0.903	0.824	0.999	0.943	0.994	0.898	—	—	—	0.926	0.941	0.912	—	—	—
BPIC13_inc	0.950	0.986	0.916	0.928	0.977	0.883	0.883	0.920	0.848	0.904	0.997	0.827	0.945	0.913	0.979	0.945	0.913	0.979	—	—	—
BPIC14	0.949	0.923	0.975	0.867	0.772	0.988	0.877	0.889	0.865	0.674	0.601	0.767	—	—	—	—	—	—	—	—	—
BPIC15_1	0.959	0.951	0.967	0.889	0.899	0.880	0.217	0.967	0.122	0.721	0.578	0.959	—	—	—	—	—	—	0.827	1.000	0.704
BPIC15_2	0.934	0.882	0.994	0.811	0.770	0.858	0.168	0.927	0.092	0.729	0.596	0.938	—	—	—	0.734	0.981	0.586	—	—	—
BPIC15_3	0.914	0.900	0.928	0.849	0.787	0.923	0.200	0.948	0.112	0.746	0.668	0.844	0.742	0.952	0.607	0.742	0.952	0.607	—	—	—
BPIC15_4	0.924	0.904	0.944	0.789	0.733	0.855	0.157	0.962	0.085	0.747	0.650	0.879	—	—	—	0.744	0.991	0.595	—	—	—
BPIC15_5	0.950	0.940	0.960	0.825	0.794	0.859	0.144	0.938	0.078	0.698	0.556	0.937	—	—	—	0.772	1.000	0.629	0.772	1.000	0.629
BPIC17	0.904	0.857	0.957	0.771	0.956	0.646	0.737	0.979	0.590	0.757	0.647	0.910	—	—	—	0.648	0.954	0.491	—	—	—
RTFMP	0.996	0.993	1.000	0.961	0.996	0.929	0.826	0.992	0.708	0.765	0.950	0.641	—	—	—	0.978	0.977	0.980	0.990	0.996	0.984
SEPSIS	0.903	0.843	0.972	0.858	0.764	0.980	0.314	0.993	0.187	0.802	0.737	0.880	—	—	—	0.630	0.922	0.479	—	—	—

F-Score (F-Sc.) measuring the accuracy, Recall (Rec.), Precision (Prec.).

alization and simplicity, but at the risk of hampering F-Score optimization. No testing was performed in this direction.

Table V compares, in percentage points, the results got by X-Processes with the results got by the best baseline discovery method, for each log. The best scores to be compared are chosen based on F-Score only, for both X-Processes and the baseline methods. For example, for the BPIC13_cp log, as ETM got the best F-Score among the baseline methods, all X-Processes scores are compared to all ETM scores (cf. Tables III and IV). The F-Score improvement reaches 30% for the log with the lowest best F-Score got by any baseline method.

Table VI shows the mean and standard deviation for F-Score based on five X-Processes runs. Despite being a probabilistic approach, X-Processes performs quite stably, with fairly low standard deviations. In addition, X-Processes got the best F-Score results when compared to the other discovery methods, even in the worst case (minimum) of five runs for all logs.

Table VII shows the execution time results. Times shown are a mean of five runs. For eight logs, execution stopped when reaching 24 hours, although they could have continued to improve F-Score results. For the other four logs that stopped evolving within 24 hours, they could also have resumed evolution if a more conservative stopping criteria had been used. Despite potentially taking a long time to run, X-Processes beat the baseline's best result with a shorter partial execution time, as shown in the third column of the table.

Fig. 7 and Fig. 8 show examples of fitness evolution from one of the X-Processes runs for one of the logs. First, Fig. 7 shows the evolution of the 20 islands in an integrated way. The convergence of the lowest, average and highest fitness values is observed for all islands with an upward trend, showing room to improve the F-Score value even further. Next, Fig. 8 shows the fitness evolution for just one of the 20 islands, where the individual evolution of an individual island becomes clearer. One can see from the charts the heuristic used to create the initial population favors the creation of individuals with reasonable initial F-Score values; in this example, around 0.4 for the worst case and around 0.8 for the best case.

Fig. 9 and Fig. 10 exemplify the process models discovered

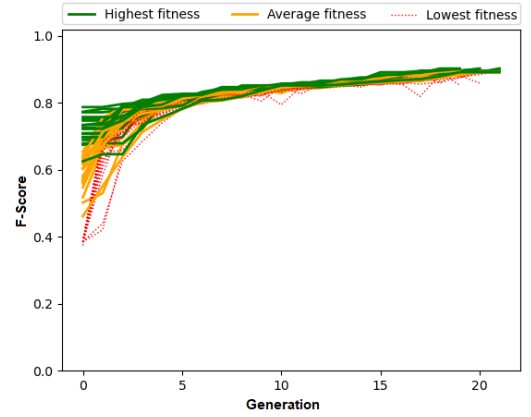


Fig. 7. X-Processes fitness evolution for the SEPSIS log (integrated view considering all the 20 islands, for the 5th run).

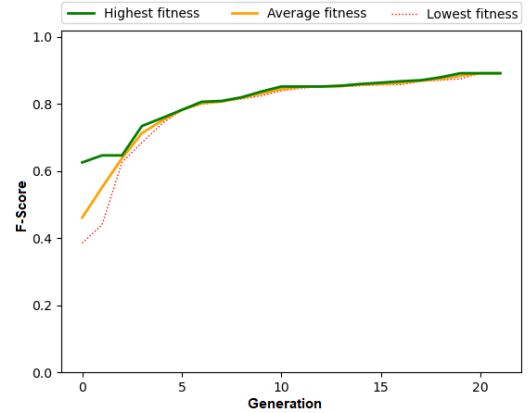


Fig. 8. X-Processes fitness evolution for the SEPSIS log (single view for one of the islands, for the 5th run).

by SM and X-Processes for the BPIC14 log. It was extracted from an ITIL Service Management tool called HP Service Manager, which records the progress of incident management. The model discovered by X-Processes exhibits a much higher recall than the one discovered by SM, as well as a higher accuracy. The precision of both models discovered is similar

TABLE IV
EXPERIMENTAL RESULTS FOR GENERALIZATION AND SIMPLICITY

Event log	X-Processes		SM		IM _F		ETM		HM ₆		S-HM ₆		FO	
	Gen.	Simp.	Gen.	Simp.	Gen.	Simp.	Gen.	Simp.	Gen.	Simp.	Gen.	Simp.	Gen.	Simp.
BPIC12	0.807	0.530	0.952	0.711	0.984	0.626	0.782	0.971	–	–	–	–	–	–
BPIC13_cp	0.867	0.733	0.944	0.793	0.938	0.882	0.790	0.449	–	–	0.361	0.705	–	–
BPIC13_inc	0.851	0.674	0.960	0.765	0.989	0.765	0.524	0.422	0.905	0.778	0.956	0.750	–	–
BPIC14	0.990	0.584	0.990	0.714	0.993	0.700	0.829	0.846	–	–	–	–	–	–
BPIC15_1	0.848	0.682	0.922	0.782	0.919	0.623	0.947	0.892	–	–	–	–	0.910	0.686
BPIC15_2	0.841	0.638	0.902	0.814	0.927	0.637	0.911	0.765	–	–	0.808	0.650	–	–
BPIC15_3	0.876	0.589	0.952	0.814	0.963	0.627	0.800	0.604	0.911	0.629	0.818	0.581	–	–
BPIC15_4	0.855	0.637	0.942	0.818	0.943	0.635	0.850	0.737	–	–	0.700	0.618	–	–
BPIC15_5	0.855	0.667	0.931	0.834	0.944	0.558	0.866	0.743	–	–	0.767	0.651	0.912	0.651
BPIC17	0.858	0.663	0.993	0.758	0.963	0.714	0.805	0.909	–	–	0.655	0.744	–	–
RTFMP	0.878	0.586	0.989	0.697	0.991	0.653	0.650	0.404	–	–	0.186	0.662	0.960	0.600
SEPSIS	0.781	0.514	0.913	0.725	0.949	0.624	0.533	0.383	–	–	0.166	0.630	–	–

Generalization (Gen.), Simplicity (Simp.).

TABLE V
X-PROCESSES IMPROVEMENT OVER BEST BASELINE METHOD

Event log	Best baseline method	F-Sc.	Rec.	Prec.	Gen.	Simp.
BPIC12	SM	30%	-19%	89%	-15%	-25%
BPIC13_cp	ETM	4%	-1%	8%	10%	63%
BPIC13_inc	S-HM ₆	1%	8%	-6%	-11%	-10%
BPIC14	IM _F	8%	4%	13%	0%	-17%
BPIC15_1	SM	8%	6%	10%	-8%	-13%
BPIC15_2	SM	15%	15%	16%	-7%	-22%
BPIC15_3	SM	8%	14%	1%	-8%	-28%
BPIC15_4	SM	17%	23%	10%	-9%	-22%
BPIC15_5	SM	15%	18%	12%	-8%	-20%
BPIC17	SM	17%	-10%	48%	-14%	-13%
RTFMP	FO	1%	0%	2%	-9%	-2%
SEPSIS	SM	5%	10%	-1%	-14%	-29%

TABLE VI
DESCRIPTIVE STATISTICAL DATA FOR F-SCORE RESULTS CONSIDERING FIVE X-PROCESSES RUNS

Event log	Minimum	Mean	Maximum	Standard Deviation
BPIC12	0.732	0.789	0.857	0.04066
BPIC13_cp	0.981	0.981	0.981	0.00000
BPIC13_inc	0.945	0.949	0.950	0.00188
BPIC14	0.938	0.944	0.949	0.00340
BPIC15_1	0.953	0.955	0.959	0.00195
BPIC15_2	0.919	0.927	0.934	0.00512
BPIC15_3	0.897	0.903	0.914	0.00594
BPIC15_4	0.904	0.914	0.924	0.00765
BPIC15_5	0.933	0.944	0.950	0.00621
BPIC17	0.873	0.881	0.904	0.01174
RTFMP	0.996	0.996	0.996	0.00003
SEPSIS	0.883	0.893	0.903	0.00819

and quite high, as neither of them exhibit *flower* behavior to achieve a high recall. Generalization of both is also similar, although it cannot be graphically visualized through the model. Although the model discovered by SM is fairly simpler, this is achieved at the expense of low recall, usually considered the most essential of the process model quality criteria.

V. CONCLUSION AND FUTURE WORK

We proposed here the process discovery method named X-Processes. X-Processes is based on genetic algorithms and

TABLE VII
X-PROCESSES EXECUTION TIME

Event log	Mean total time (hh:mm:ss)	Mean time to beat the baseline's best result (hh:mm:ss)
BPIC12	24:00:00	13:04:10
BPIC13_cp	00:07:03	00:00:28
BPIC13_inc	05:38:51	01:41:40
BPIC14	24:00:00	04:32:57
BPIC15_1	10:33:39	00:13:51
BPIC15_2	24:00:00	01:04:04
BPIC15_3	24:00:00	02:09:19
BPIC15_4	24:00:00	00:37:49
BPIC15_5	24:00:00	00:38:30
BPIC17	24:00:00	02:04:07
RTFMP	02:43:34	00:21:25
SEPSIS	24:00:00	15:54:15

the island model. Our results show X-Processes can discover sound and highly accurate process models, i.e., with high recall and high precision combined, surpassing baselines. In fact, X-processes achieves accuracy above 0.9 for 11 out of 12 tested logs. As a limitation, X-Processes offers high accuracy at the expense of low generalization and simplicity, as well as high execution time. While execution time may be a relevant factor when discovering a number of process models, high accuracy is much more relevant when the quality of a particular process model to be discovered is paramount. After using less accurate discovery methods to efficiently explore different analysis scenarios of a business process, one can certainly benefit from using X-Processes to discover a highly accurate model for a specific scenario even having to wait a few hours.

There are several possibilities for extending X-Processes. Its chromosomal representation aligned with the inherent flexibility of its fitness function allow us to advance in: (i) incorporating the generalization and simplicity criteria in the optimization process to improve quality, and (ii) developing our own code implementation taking advantage of the binary chromosome representation to perform the soundness check and quality measure calculation to improve efficiency. Furthermore, to build a search space better suitable to finding process

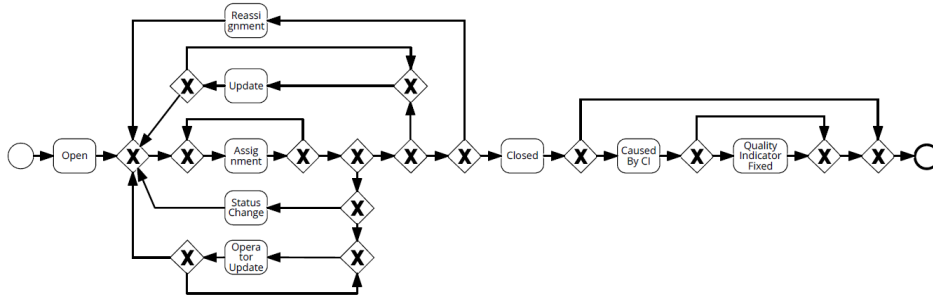


Fig. 9. Process model discovered by SM from the BPIC14 log.

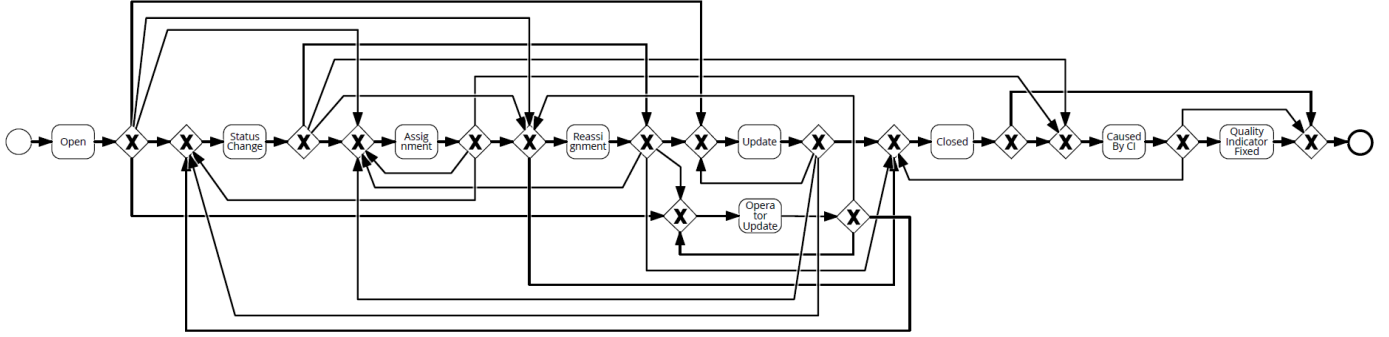


Fig. 10. Process model discovered by X-Processes from the BPIC14 log.

models with greater simplicity, the chromosome representation should be extended to represent models with composite gateways. Finally, addressing infrequent behavior can help improve quality, including in terms of generalization.

VI. ACKNOWLEDGMENTS

We thank Adriano Augusto and Marlon Dumas for sending us the filtered logs for running our experiment, and Joon Hyuk Kim for his support in implementing the islands model.

REFERENCES

- [1] W. M. P. van der Aalst, *Process Mining – Data Science in Action*. Springer, 2nd ed., 2016.
- [2] W. M. P. van der Aalst, T. Weijters, and L. Maruster, “Workflow mining: Discovering process models from event logs,” *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.
- [3] A. J. M. M. Weijters and J. T. S. Ribeiro, “Flexible heuristics miner (FHM),” in *Proc. of the IEEE Symp. on Comput. Intell. and Data Min.*, pp. 310–317, IEEE, 2011.
- [4] S. K. L. M. vanden Broucke and J. D. Weerdt, “Fodina: A robust and flexible heuristic process discovery technique,” *Decis. Support Syst.*, vol. 100, pp. 109–118, 2017.
- [5] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering block-structured process models from event logs containing infrequent behaviour,” in *Proc. of the 9th Int’l Workshop on Bus. Process Intell.*, vol. 171, pp. 66–78, Springer, 2013.
- [6] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, and G. Bruno, “Automated discovery of structured process models: Discover structured vs. discover and structure,” in *Proc. of the 35th Int’l Conf. on Conceptual Model.*, pp. 313–329, Springer, 2016.
- [7] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, and A. Polyvyanyy, “Split miner: automated discovery of accurate and simple business process models from event logs,” *Knowl. Inf. Syst.*, vol. 59, no. 2, pp. 251–284, 2019.
- [8] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst, “Genetic process mining: An experimental evaluation,” *Data Min. Knowl. Discovery*, vol. 14, no. 2, pp. 245–304, 2007.
- [9] C. Bratosin, N. Sidorova, and W. M. P. van der Aalst, “Distributed genetic process mining using sampling,” in *Proc. of the 11th Int’l Conf. on Parallel Comput. Technol.*, pp. 224–237, Springer, 2011.
- [10] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, “Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity,” *Int’l J. of Cooperative Inf. Syst.*, vol. 23, no. 1, 2014.
- [11] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn, “Soundness of workflow nets: classification, decidability, and analysis,” *Formal Aspects of Comput.*, vol. 23, no. 3, pp. 333–363, 2011.
- [12] K. A. De Jong, *Evolutionary Computation: A Unified Approach*. MIT press, 2006.
- [13] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms*. Springer, 2008.
- [14] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst, “Conformance checking using cost-based fitness analysis,” in *Proc. of the 15th IEEE Int’l Enterp. Distrib. Object Comput. Conf.*, pp. 55–64, IEEE, 2011.
- [15] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. M. P. van der Aalst, “Measuring precision of modeled behavior,” *Inf. Syst. and e-Business Manag.*, vol. 13, no. 1, pp. 37–67, 2015.
- [16] F. R. Blum, “Metrics in process discovery,” Tech. Rep. Technical Report TR/DCC-2015-6, Computer Science Dept., University of Chile, 2015.
- [17] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, “A genetic algorithm for discovering process trees,” in *Proc. of the IEEE Congress on Evol. Comput.*, pp. 1–8, IEEE, 2012.
- [18] D. Whitley, S. Rana, and R. Heckendorn, “Island model genetic algorithms and linearly separable problems,” in *Proc. of the AISB Int’l Workshop on Evolutionary Computing*, pp. 109–125, Springer, 1997.
- [19] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, “A genetic algorithm for discovering process trees,” in *Proc. of the IEEE Congress on Evol. Comput.*, pp. 1–8, IEEE, 2012.
- [20] D. Hauschildt, H. M. W. Verbeek, and W. M. P. van der Aalst, “WOFLAN: a petri-net-based workflow analyzer,” Tech. Rep. Computing Science Reports 97/12, Eindhoven University of Technology, 1997.
- [21] D. Bhandari and N. R. Pal, “Directed mutation in genetic algorithms,” *Inf. Sci.*, vol. 79, no. 3–4, pp. 251–270, 1994.