Mining Frequent Patterns in Process Models

David Chapela-Campa[a,*], Manuel Mucientes[a], Manuel Lama[a]

[a]*Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)*
Universidade de Santiago de Compostela. Santiago de Compostela, Spain

Abstract

Process mining has emerged as a way to analyze the behavior of an organization by extracting knowledge from event logs and by offering techniques to discover, monitor and enhance real processes. In the discovery of process models, retrieving a complex one, i.e., a hardly readable process model, can hinder the extraction of information. Even in well-structured process models, there is information that cannot be obtained with the current techniques. In this paper, we present WoMine, an algorithm to retrieve frequent behavioural patterns from the model. Our approach searches in process models extracting structures with sequences, selections, parallels and loops, which are frequently executed in the logs. This proposal has been validated with a set of process models, including some from BPI Challenges, and compared with the state of the art techniques. Experiments have validated that WoMine can find all types of patterns, extracting information that cannot be mined with the state of the art techniques.

Keywords: Frequent pattern mining, Process mining, Process discovery

1. Introduction

With the explosion of process-related data, the behavioural analysis and study of business processes has become more popular. Process mining offers techniques to discover, monitor and enhance real processes by extracting knowledge from event logs, allowing to understand *what is really happening in a business process*, and not *what we think is going on* [25]. Nevertheless, there are scenarios —highly complex process models— where process discovery techniques are not able to provide enough intelligible information to make the process model understandable to users.

The starting point of process discovery is a log, i.e., a set of traces. Each trace contains the sequence of events which have been executed in an instance of the process. An event corresponds with an execution of an activity of the process, with information about the execution as the start and end times. With this information, the process discovery algorithms build a directed graph, or process model, with the relations between the activities based on the log. This process model represents the execution of the process based in the behavior in the log.

There are four quality dimensions to measure *how good* a process model is: *fitness replay*, which quantifies the extent to which the discovered model can accurately reproduce the cases recorded in the log; *precision*, which quantifies the fraction of the behavior allowed by the model which is not seen in the event log; *generalization*, which assesses the extent to which the resulting model will be able to reproduce future behavior of the process; and *simplicity*, which represents the structual complexity of the model [4]. Regarding the latter, discovering a complex process model, i.e., a hardly readable process model, can totally hinder its quality [7] making difficult the retrieval of behavioural information. Different techniques have been proposed to tackle this problem: the simplification of already mined models [7, 9], the search of simpler structures in the logs [17, 19, 24], or the clusterization of the log into smaller and more homogeneous subsets of traces to discover different models within the same process [12, 13, 22]. Although these techniques improve the understandability of the process models, for real processes the model structure remains complex, being difficult to understand by users.

[*] Corresponding author

Email addresses: david.chapela@usc.es (David Chapela-Campa), manuel.mucientes@usc.es (Manuel Mucientes), manuel.lama@usc.es (Manuel Lama)

As an alternative to these techniques, there exist some proposals whose aim is to extract structures —or subprocesses— within the model which are relevant to describe the process model. In these approaches, the relevance of a structure is measured as: *i)* the total number of executions [17, 24], e.g., a structure executed 1,000 times; or *ii)* its high repetition in the traces of the log [3, 11], e.g., a subprocess which appears most of the times the process is executed. In this paper, we will focus in the search of frequent behavioural patterns of the second type. The extraction of these frequent structures is useful in both highly complex and well-structured models. In complex models, it allows to abstract from all the behaviour and focus on relevant structures. Additionally, the application of these techniques in well-structured process models retrieves frequent subprocesses which can be, for instance, the objective of optimizations due to its frequent execution within the process. The knowledge obtained by extracting these frequent patterns is valuable in many fields. For instance, in e-learning, the interactions of the students with the learning management system can be registered in order to reconstruct their behaviour during the subject [33], i.e., to reconstruct the learning path followed by the students. The extraction of frequent patterns from these learning paths —processes— can help teachers to improve the learning design of the subject, enabling its adaptation to the students behaviour. It can also reveal behavioral patterns that should not be happening. In addition, for other business processes like, for example, call centers where the objective is to retain the customers, the discovery of frequent behaviours can be decision-making. In this domain, process models tend to contain numerous choices and loops, where frequent structures can show a possible behavior which leads to retain customers. This knowledge can be used to plan new strategies in order to reduce the number of clients who drop out, by exploiting the paths that lead to retain customers, or avoiding those which end in dropping out.

In this paper we present WoMine, an algorithm to mine frequent patterns from a process model, measuring their frequency in the instances of the log. The main novelty of WoMine, which is based on the *w-find* algorithm [11], is that it can detect frequent patterns with all type of structures —even n-length cycles, very common structures in real processes. It can also ensure which traces are compliant with the frequent pattern in a percentage over a threshold. Furthermore, WoMine is robust w.r.t. the quality of the mined models with which it works, i.e., its results do not depend highly on the fitness replay and precision of the mined models. The algorithm has been tested with 20 synthetic process models ranging from 20 to 30 unique activities, and containing loops, parallelisms, selections, etc. Experiments have been also run with 12 real complex logs of the Business Process Intelligence Challenges.

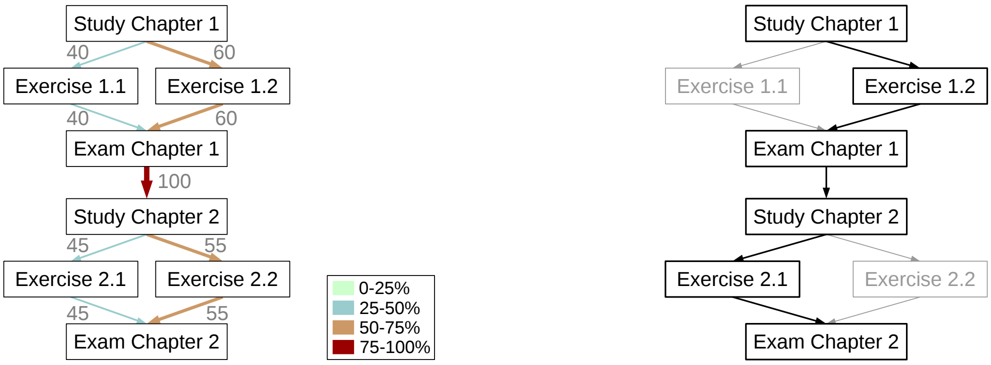
The remainder of this paper is structured as follows. Section 2 introduces the algorithms related with the purpose of this paper. The required background of the paper is introduced in Section 3. Section 4 presents the main structure of WoMine, followed by detailed explanations in Sections 5 and 6. Finally, Section 7 describes an evaluation of the approach, and Section 8 summarizes the conclusions of the paper.

2. Related Work

A simple and popular technique to detect frequent structures in a process model is the use of *heat maps*, which can be found in applications like DISCO [14]. It provides a simple technique which can retrieve the frequent structures of a process model considering the individual frequency of each arc. Other techniques check the frequency of each pattern taking into account all the structure, and not the individual frequency. These approaches, under the frequent pattern mining field [15], can build frequent patterns based just on the logs, searching in them for frequent sequences of activities [16, 21, 36]. Improving this search, episode mining techniques focus their search in frequent, and more complex structures such as parallels [17, 19]. With a different approach, the *w*-find algorithm [11] uses the process model to build the patterns, checking their frequency in the logs. Extending these mining techniques, the local process mining approach of Niek Tax et al. [24] discovers frequent patterns from the logs providing support to loops. Finally, in [3] tree structured patterns in the XML structure of the XES[1] logs are searched. Nevertheless, as we will show in this section, all these techniques fail to measure the frequency of a pattern in some cases, and specially when the model presents loops or optional activities[2].

[1] XES is an XML-based standard for event logs. Its purpose is to provide a generally-acknowledged format for the interchange of event log data between tools and application domains (`http://www.xes-standard.org/`).

[2] In this paper we will refer as optional activities to the activities of a selection (choice) where one of the branches has no activities, leaving the other as optional.



(a) Heat Maps example: C-net with the arcs highlighted depending on their absolute frequency.

(b) WoMine example: Model with a frequent pattern (40%) highlighted.

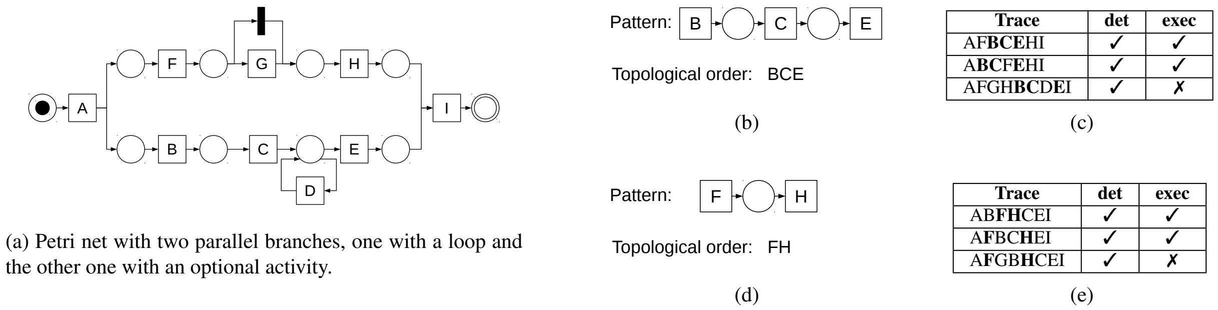
Figure 1: Process model of a simple process in the education domain.

The heat maps approach performs a highlight of the arcs and activities of a process model depending on their individual frequency, i.e., the number of executions. To obtain the frequent patterns of a model using heat maps, the arcs which frequency exceeds a defined threshold are retrieved. The problem is that the individual frequency does not consider the causality between activities. A highlighted structure can have all its arcs individually frequent, i.e., each arc is executed individually in a percentage of traces considered frequent, but it does not have to be necessarily frequent, i.e., the highlighted structure is not executed completely in a percentage of traces considered frequent.

As an example, the process model in Fig. 1 is provided. This model represents the behavior performed by the students in an educational process, e.g. a course. In this course, the students must study a chapter, choose between two exercises and take an exam. This process is repeated for each chapter. If we prune on 50% —retrieve the arcs with a frequency over 50— the model in Fig. 1a, the path (*'Study Chapter 1'*, *'Exercise 1.2'*, *'Exam Chapter 1'*, *'Study Chapter 2'*, *'Exercise 2.2'*, *'Exam Chapter 2'*), is obtained. Conversely, if WoMine searches with a threshold of 40%, this behavioural structure is not among the results —because the individual arcs are frequent, but the sequence is not, i.e., the students solving *'Exercise 1.2'* and those doing *'Exercise 2.2'* are not the same. Instead, with WoMine, the structure of Fig. 1b is obtained, providing the information that in 40 traces of 100, the students select exercises 1.2 and 2.1. Besides, the 88.88% —40 out of 45— of the students who choose the exercise 2.1 came from exercise 1.2. This behaviour can hint a predilection in the students who solve the exercise 1.2 to choose the exercise 2.1.

As can be seen, heat maps cannot find frequent structures to identify real common behaviour in processes. There are approaches that retrieve frequent patterns measuring the frequency of the whole structure [2, 3, 21, 17, 19]. Some of these techniques are based on sequential pattern mining (SPM), and search subsequences of activities with a high frequency in large sequences [2, 21]. One of the first approaches was proposed by Agrawal et al. [2], preceded by techniques to retrieve association rules between itemsets [1]. Most of the designed sequential pattern mining techniques present an expensive candidate generation and testing, which induces a long runtime in complex cases. To compare the features of SPM against other discussed in this paper, we will use PrefixSpan [21], which performs a pattern-growth mining with a projection to a database based in frequent prefixes, instead of considering all the possible occurrences of frequent subsequences. The main drawback of the sequential pattern mining based approaches is the simplicity of the patterns mined —sequences of activities. Structures like concurrences or selections are treated as different sequences depending on the order of the activities. Also, in a retrieved frequent sequence, the execution of the i-th activity is not ensured to be caused by the $i-1$-th activity in all the occurrences in the log —SPM only checks if the activities of the sequence appear in the trace in the same order.

The episode mining based approaches appear to improve the results of SPM. The first reference to episode mining is done by Mannila et al. [19]. An episode is a collection of activities occurring close to each other. Their algorithm uses *windows* with a predefined width to extract frequent episodes, being able to detect episodes with sequences and concurrency. In this approach, an episode is frequent when it appears in many different windows. Based on this,



(a) Petri net with two parallel branches, one with a loop and the other one with an optional activity.

(b)

Trace	det	exec
AF**BCE**HI	✓	✓
A**BC**F**E**HI	✓	✓
AFGH**BC**D**E**I	✓	✗

(c)

(d)

Trace	det	exec
AB**FH**CEI	✓	✓
A**F**BC**H**EI	✓	✓
A**F**GB**H**CEI	✓	✗

(e)

Figure 2: Model and examples to show the problems with the use of the topological order of a pattern to measure its frequency.

Leemans et al. have designed an algorithm [17] to extract association rules from frequent episodes measuring their frequency with the instances of the process, instead of using windows in the activity sequences. For instance, an episode with a frequency of 50% has been executed in a half of the recorded traces of the process model. One of the drawbacks of episode mining based techniques, which this paper tries to tackle, is that its search is not based in the model and, thus, it does not take advantage of the relation among activities presented in it. For the same frequent behaviour, the algorithm extracts various patterns with the same activities, but with different relations among them, making difficult the extraction of information.

To take advantage of the knowledge generated by the discovery algorithm, the frequent structures can be built based on the process model. Greco et al. have developed an algorithm which mines frequent patterns in process models, *w*-find [11] —the algorithm on which WoMine is based. This approach uses the model to build patterns compliant to the model, reducing the search space and measuring their frequency with the set of instances of the log. Thus, using an a priori approach which grows the frequent patterns, this algorithm retrieves structures from the model, which are executed in a high percentage of the instances. The drawback of this approach is the simplicity of the mined structures. The patterns cannot contain selections and, thus, it will never retrieve patterns with loops.

Previous techniques —SPM, episode mining and *w*-find— measure the frequency of the structures checking for the topological order of the structure in the traces. But this method is not able to detect correctly the execution of a pattern when loops or optional activities appear. For a better understanding the example shown in Fig. 2 is presented. The pattern shown in Fig. 2b represents the bottom branch of the model in Fig. 2a without the loop. In Fig. 2c three examples of traces are presented. *Exec* stands for the real execution of the pattern in the trace, while *det* is positive when the topological order appears in the trace. A foreign activity in the middle of the topological order does not invalid the detection because it can be from the execution of the other parallel branch —trace 2—. This pattern is correctly executed and detected in the two first traces. However, when the loop is executed —third trace— disrupting the execution of the pattern, its topological order still appears in the trace and, thus, the pattern is detected incorrectly. The same problem occurs with the pattern shown in Fig. 2d, which is disrupted in the third trace with the execution of G in the middle.

Another approach, called local process mining, is presented in [24]. In this approach a discovery of simple process models representing frequent behaviour is performed, instead of a complex process model representing all the behaviour. In local process mining, tree models are built by performing an iterative growing process, starting with single activities, and adding different relations with other activities. The evaluation of the frequency is done with an alignment-based method which, starting with an initial marking, considers that the model is executed when the final marking is reached. This leads to one of the drawbacks of this technique; when a model contains loops or selections, the evaluation counts the model as executed even if the loop, or a choice of the selection, has not been executed in that trace.

Finally, the approach presented by Bui et al. [3] uses logs in a XES format, and performs a search of tree structures in the XML structure of the log. This approach builds a tree with the characteristics of each trace, and uses tree mining techniques to search frequent structures. The information retrieved is a frequent subset of activities and common characteristics of the XES structure. A drawback of this approach, as with SPM algorithms, is that the retrieved patterns can only ensure the order of the activities, but not the relation between them.

In summary, the extraction of frequent patterns could be done highlighting the frequent elements —heat maps— of the process model and pruning them, but without ensuring the real frequency of the results. Sequential pattern mining could be used to retrieve real frequent patterns but the structures are limited to sequences, and it only ensures the precedence between the activities. The approaches based on episode mining [17, 19] and frequent pattern mining [11] retrieves structures that are still simple for real processes, and the measure of the frequency presents problems when the process model has loops or optional activities. Finally, local process mining provides similar results, with the addition of loops and choices to the extracted models. Other types of search techniques have also been applied like mining tree structures but, as sequential pattern mining, the activities of the retrieved patterns have only sequences.

As far as we know, the *w*-find is the only algorithm that searches substructures in the process model, checking the frequency in the traces of the log. The algorithm presented in this paper, WoMine, realizes an a priori search based on the *w*-find search, being able to get frequent subprocesses in models with loops and optional activities, ensuring the frequency of each pattern and retrieving structures with sequences, parallels, selections and loops. A comparison between these algorithms is presented in Table 1.

3. Preliminaries

In this paper, we will represent the examples with place/transition Petri nets [8] due to its higher comprehensibility, and the easiness to explain the behaviour of the execution. A Petri net is a directed graph composed by two kind of nodes: places and transitions —circles and boxes, respectively—, and where the arcs connect two nodes of different type. A transition is said to be *enabled* when all the places of its inputs —with an arc to it— contain, at least, a token —represented by a dot. The execution of an enabled transition consumes a token from each place of its inputs, and generates a token in each place of its outputs —places with an arc from the transition to them. In this way, a Petri net allows to represent the behaviour in a process showing the relation between its activities. Nevertheless, our algorithm represents the process with a Causal net (Def. 1).

Definition 1 (Causal net [27]). A Causal net (C-net) is a tuple $C = (A, a_i, a_o, D, I, O)$ where:

- A is a finite set of activities;
- $a_i \in A$ is the start activity;
- $a_o \in A$ is the end activity;
- $D \subseteq A \times A$ is the dependency relation,

	Mine from model	Expl. proc. instances	Mine sequences	Mine parallels	Mine choices	Mine loops
Sequential Pattern Mining - Based Algorithms [21]	-	-	+	-	-	-
Mannila's Episode mining [19]	-	-	+	+	-	-
Bui's Tree Mining [3]	-	+	+	-	-	-
Leemans' Episode discovery [17]	-	+	+	+	-	-
w-find [11]	+	+	+	+	-	-
Tax's Local Process Model [24]	-	+	+	+	±	±
WoMine (this publication)	**+**	**+**	**+**	**+**	**+**	**+**

Table 1: Feature comparison of discussed algorithms. *'Mine from model'* is marked if the algorithm uses the process model to retrieve the patterns, basing the search on the relations in the model. *'Expl. proc. instances'* indicates if the algorithm uses the traces of the log to measure the frequency of the patterns being a 100% the apparition in all traces. *'Mine sequences'*, *'parallels'*, *'choices'* and *'loops'* indicate if the algorithm retrieves frequent patterns with sequences, parallels, choices or loops, respectively. Finally *'+'* stands for a complete support to the feature, *'-'* stands for a non support and *'±'* stands for a partial support to the feature for the purpose of this paper.

- $AS = \{X \subseteq \mathcal{P}(A) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$;[3]
- $I \in A \rightarrow AS$ defines the set of possible input bindings per activity;
- $O \in A \rightarrow AS$ defines the set of possible output bindings per activity,

such that:

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup_{as \in I(a_2)} as\}$;
- $D = \{(a_1, a_2) \in A \times A \mid a_2 \in \bigcup_{as \in I(a_1)} as\}$;
- $\{a_i\} = \{a \in A \mid I(a) = \{\emptyset\}\}$;
- $\{a_o\} = \{a \in A \mid O(a) = \{\emptyset\}\}$;
- all activities in the graph (A, D) are on a path from a_i to a_o.

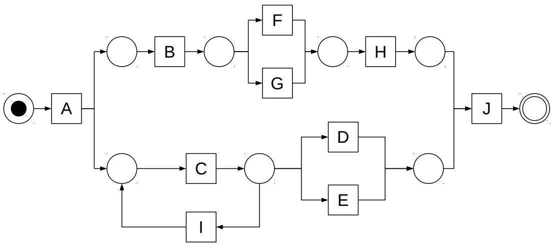
Fig. 3c shows the Causal net of the process model from Fig. 3a.

Definition 2 (Trace). Let A be the set of activities of a process model, and ε an event —the execution of an activity $\alpha \in A$. A trace is a list (sequence) $\tau = \varepsilon_1, ..., \varepsilon_n$ of events ε_i occurring at a time index i relative to the other events in τ. Each trace corresponds to an execution of the process, i.e., a process instance. As an example, Fig 3b shows a trace that corresponds to an execution of the process model of Fig. 3a.

Definition 3 (Log). An event log $L = [\tau_1, ..., \tau_m]$ is a multiset of traces τ_i. In this simple definition, events only specify the name of the activity, but usually, event logs store more information as timestamps, resources, etc.

Definition 4 (Pattern). Let $C = (A, a_i, a_o, D, I, O)$ be a C-net representing a process model M. A connected subgraph represented by the C-net $P = (A', A'_i, A'_o, D', I', O')$, where $A'_i \subseteq A'$ and $A'_o \subseteq A'$ represent respectively the start and end activities, is a pattern of M if and only if:

- $A' \subseteq A$;
- $D' \subseteq D$;



(a) Petri net, with XOR and AND structures.

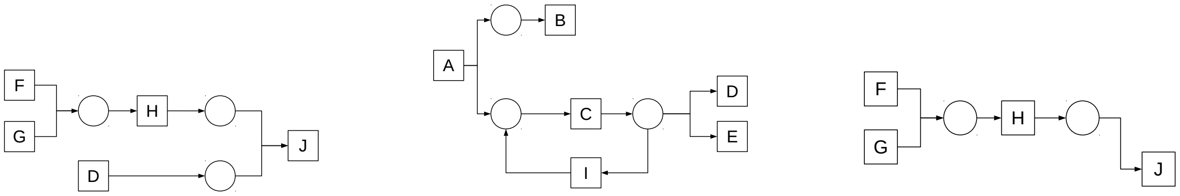
A C B I C I C F H E J

(b) Trace of the process model in Fig. 3a.

Activity	I(Activity)	O(Activity)
A	{}	{{B,C}}
B	{{A}}	{{F},{G}}
C	{{A},{I}}	{{D},{E},{I}}
D	{{C}}	{{J}}
E	{{C}}	{{J}}
F	{{B}}	{{H}}
G	{{B}}	{{H}}
H	{{F},{G}}	{{J}}
I	{{C}}	{{C}}
J	{{H,D},{H,E}}	{}

(c) Causal net connections of the process model in Fig. 3a.

Figure 3: Example to show the internal representation of the process models in WoMine. The inputs of activity J are composed by two paths or choices. One is the tuple H and D, and the other one is the tuple H and E. As can be seen, each subset in the set of inputs (I(J)) corresponds to a possible path in the inputs of J.



(a) Valid pattern with a selection and a parallel.

(b) Valid pattern with a parallel, a selection and a loop.

(c) Invalid pattern. Activity `J` has some incomplete input combinations.

Figure 4: Examples of valid and invalid patterns of the process model shown in Fig. 3. All activities have as connections a subset of their connections in the causal net of Table 3c. For instance, $I'(J)$ is equal to `{{H,D}}`, which is a subset of $I(J)$, `{{H,D}, {H,E}}`. This makes the structure a valid pattern. Meanwhile, the structure shown in Fig. 4c is a wrong pattern, because activity `J` has some incomplete input combinations —`{{H}}` $\nsubseteq$ `{{H,D}, {H,E}}`.

- for any $\alpha \in A'$: $I'(\alpha) \subseteq I(\alpha), O'(\alpha) \subseteq O(\alpha)$

A *pattern* (Def. 4) is a subgraph of the process model that represents the behaviour of a part of the process. For each activity α in the pattern, its inputs, $I'(\alpha)$, must be a subset of $I(\alpha)$ in the model it belongs to; and the outputs, $O'(\alpha)$, must be also a subset of $O(\alpha)$ in the model. This ensures that a pattern has not a partial parallel connection. Fig. 4 shows some examples of valid and invalid patterns.

Definition 5 (Simple pattern). A pattern $P = (A', A'_i, A'_o, D', I', O')$ is a simple pattern if and only if, for all activities $\alpha \in A'$:

- $[\exists! \Phi \in I'(\alpha)\colon\ \Phi \nsubseteq R^+_\alpha] \vee [\forall \Phi \in I'(\alpha)\colon\ \Phi \subseteq R^+(\alpha)]$;
- $[\exists! \Theta \in O'(\alpha)\colon\ \Theta \nsubseteq R^-_\alpha] \vee [\forall \Theta \in O'(\alpha)\colon\ \Theta \subseteq R^-(\alpha)]$

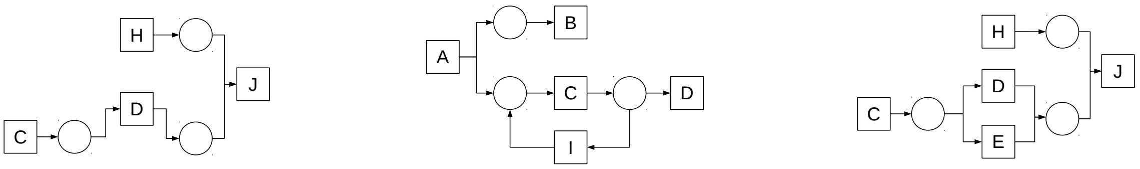
Being R^+_α the set of successors[4] of an activity α, and R^-_α the set of predecessors [5] of an activity α.

The Simple Patterns (Def. 5) are those patterns which behaviour can be executed, entirely, in one instance. If an activity has a selection, it must be able to execute each path in the same instance. For this, the inputs of each activity α must have all activities reachable from α except, at most, the activities of one path. The outputs present the same constraint, but in this case they must reach α, not be reachable by α. Fig. 5 shows two valid simple patterns and an invalid one.

Definition 6 (Minimal pattern, *M*-pattern). Each activity of the process model belongs to, at least, one minimal pattern. The *M*-pattern of an activity α corresponds to the closure of α, i.e., the structure that is going to be executed when α is executed. An exception is made with parallel structures: if α has a parallel in its inputs or outputs, there must be an *M*-pattern containing each parallel path.

Given a C-net $C = (A, a_i, a_o, D, I, O)$ representing a process model M and an activity $\alpha' \in A$, a pattern $P = (A', A'_i, A'_o, D', I', O')$ is a Minimal Pattern of α' if and only if is a maximum simple pattern containing α' and fulfilling the following rules:

- if $|I(\alpha')| > 1$ then $[I'(\alpha') = \emptyset] \vee [|I'(\alpha')| = 1, \Phi \in I'(\alpha')\colon\ |\Phi| > 1]$;
- if $|O(\alpha')| > 1$ then $[O'(\alpha') = \emptyset] \vee [|O'(\alpha')| = 1, \Theta \in O'(\alpha')\colon\ |\Theta| > 1]$;
- $\forall \alpha \in R^+_{\alpha'}\colon$ if $|O(\alpha)| \neq 1$ then $O'(\alpha) = \emptyset$;
- $\forall \alpha \in R^-_{\alpha'}\colon$ if $|I(\alpha)| \neq 1$ then $I'(\alpha) = \emptyset$;
- $\forall \alpha \in A', \alpha \neq \alpha', \alpha \notin (R^+_{\alpha'} \bigcup R^-_{\alpha'})\colon$ if $|I(\alpha)| \neq 1$ then $I'(\alpha) = \emptyset$, and if $|O(\alpha)| \neq 1$ then $O'(\alpha) = \emptyset$

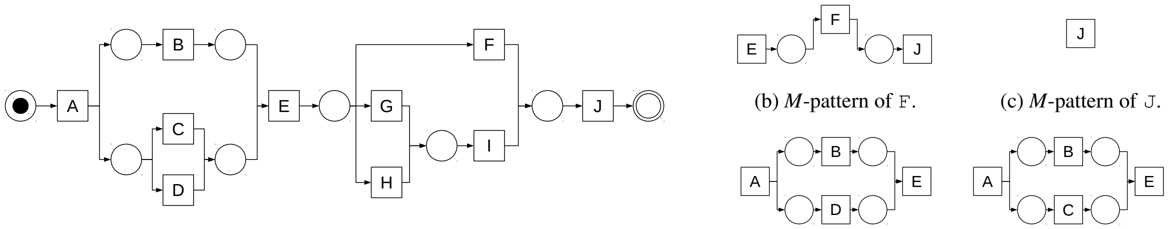


(a) Valid simple pattern. The pattern is executed in the instance `[C D H J]`.

(b) Valid simple pattern with a loop. In the output paths of `C` ({`D`} and {`I`}), the only non predecessor is the path of {`D`}. The same happens in the inputs. The pattern is executed in the instance `[A B C I C D]`.

(c) Invalid simple pattern. Activities `D` and `E` cannot be in an instance of the pattern at the same time.

Figure 5: Examples of valid and invalid simple patterns of the process model shown in Fig. 3.



(b) M-pattern of `F`.

(c) M-pattern of `J`.

(a) Petri net of the process model.

(d) M-patterns of `A`.

Figure 6: A process model and three examples of M-patterns.

In WoMine each activity α' is associated, at least, to an M-pattern. The M-patterns of an activity α' are obtained through an expansion process that starts in α' and continues through its inputs and outputs fulfilling the following rules: *i)* the process will not expand through the inputs of α' with size 1 and being part of a selection; *ii)* the same stands for the outputs of α'; *iii)* for all the successors of α' the expansion stops if the outputs are formed by a selection; *iv)* the same stands for the inputs of the predecessors of α'; *v)* finally, the process does not expand either through the inputs or outputs of the activities not fitting the previous constraints if those are formed by an XOR structure in the model.

Fig. 6 shows some M-patterns of a model. Fig. 6b shows the M-pattern of `F`: the process starts in `F` and expands the M-pattern through `F` inputs and outputs, because both are formed by only one path. The backwards expansion stops in `E` because its inputs are part of a selection. Fig. 6c depicts the M-pattern of `J`. It is formed only by itself, because its inputs are part of a selection and its outputs are empty. Finally, Fig. 6d presents the two M-patterns of `A`. As `A` is an AND-split with a selection, two M-patterns are created, each one with one of the possible paths.

Definition 7 (Candidate arcs). Let $C = (A, a_i, a_o, D, I, O)$ be a causal net representing a process model M. An arc $\langle \alpha_i \to \alpha_j \rangle\colon \alpha_i, \alpha_j \in A$ is part of the $A^<$ set, i.e., a candidate arc, if and only if:

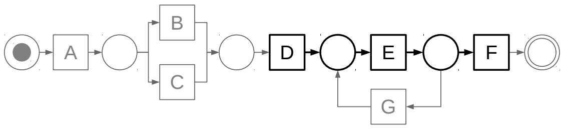
- $O(\alpha_i) = \{\Theta \in AS \mid \Theta = \{\alpha_j\} \lor \alpha_j \notin \Theta\}$
- $I(\alpha_j) = \{\Phi \in AS \mid \Phi = \{\alpha_i\} \lor \alpha_i \notin \Phi\}$

The set of candidate arcs, or $A^<$, is a subset of the arcs in the model which are not part of an AND structure. For instance, all arcs of Fig. 6a, but those starting in `A` or ending in `E`, are included in the $A^<$ set.

[3] $\mathcal{P}(A) = \{A' \mid A' \subseteq A\}$ is the powerset of A. Hence, elements of AS are *sets of sets* of activities.

[4] The successors of an activity α are the activities with a path from α to them, e.g., the successors of `B` in Fig. 3a are `F`, `G`, `H` and `J`.

[5] The predecessors of an activity α are the activities with a path from them to α, e.g., the predecessors of `C` in Fig. 3a are `C`, `I` and `A`.



Trace:
`A B D E G E F`

Executed arcs:
$\langle A \rightarrow B\rangle, \langle B \rightarrow D\rangle,$
$\langle D \rightarrow E\rangle, \langle E \rightarrow G\rangle,$
$\langle G \rightarrow E\rangle, \langle E \rightarrow F\rangle$

Figure 7: Process model with a pattern highlighted and a trace that is not compliant with the pattern due to the execution of the loop.

Definition 8 (Compliance). Given a trace $\tau \in L$ and a simple pattern *SP* belonging to the process model, the trace is compliant with *SP*, denoted as $SP \vdash \tau$, when the replay of the trace in the process model contains the replay of the pattern, i.e., all the arcs and activities of *SP* are executed in a correct order, and each activity fires the execution of its output activities in the pattern.

To check the compliance it is important that, while the pattern is being executed, each of its activities triggers only the execution of its outputs. There are cases where all the arcs and activities of the pattern are executed in a correct order, but the execution of the pattern is disrupted in the middle of it (Fig. 7). In the trace, the activities and arcs of the pattern are executed in a proper order, but the trace is not compliant with the pattern because the sequence `D-E-F` is disrupted with the execution of `G`.

Definition 9 (Frequency of pattern and simple pattern). Let *L* be the set of traces of the process log. The frequency of a simple pattern *SP* is the number of traces compliant with *SP* divided by the size of the log:

$$freq(SP) = \frac{|\{\tau \in L \colon SP \vdash \tau\}|}{|L|} \tag{1}$$

And the frequency of a pattern *P* is the minimum frequency of the simple patterns it represents:

$$freq(P) = \min_{\forall SP \in P} freq(SP) \tag{2}$$

Definition 10 (Frequent Pattern). Given a frequency threshold $\sigma \in \mathbb{R} \colon 0 < \sigma \leq 1$, a pattern *P* is a frequent pattern if and only if $freq(P) \geq \sigma$.

4. WoMine

Given a process model and a set of instances, i.e., executions of the process, the objective is to extract the subgraphs of the process model that are executed in a percentage of the traces over a threshold. A simple approach might be a brute-force algorithm, checking the frequency of every existent subgraph inside the process model, and retrieving the frequent ones. The computational cost of this approach makes it a non-viable option. The algorithm presented in this paper performs an a priori search[6] [11] starting with the frequent minimal patterns (Def. 6) of the model. In this search, there is an expansion stage done in two ways: *i)* adding frequent *M*-patterns not contained in the current pattern, and *ii)* adding frequent arcs of the $A^{<}$ set (Def. 7). This expansion is followed by a pruning strategy that verifies the downward-closure property of support [1] —also known as anti-monotonicity. This property ensures that if a pattern appears in a given number of traces, all patterns containing it will appear, at most, in the same number of

[6] An a priori search uses the previous knowledge, i.e., the a priori knowledge. It reduces the search space by pruning the exploration of those paths that will not finish in a valuable result.

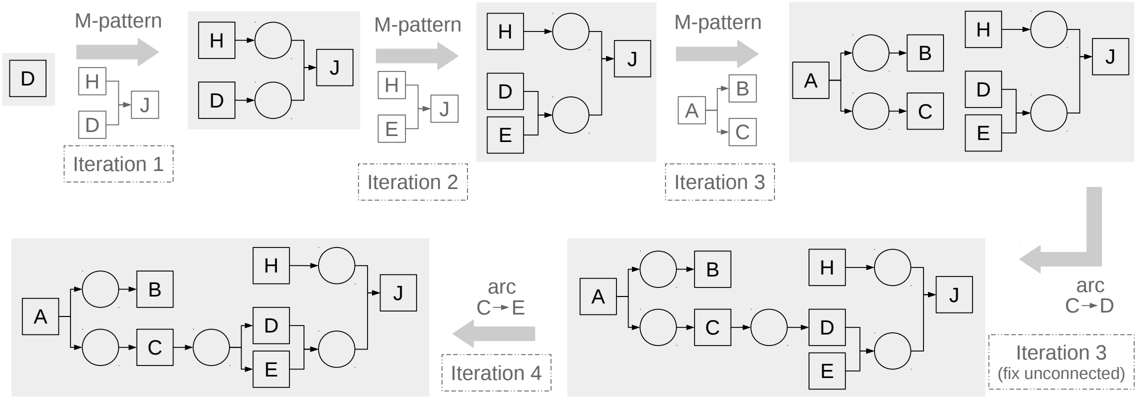


Figure 8: Example of the expansion of a pattern in 4 iterations. The process starts with the M-pattern of `D`, being expanded with an M-pattern of `J` in the first iteration. In the second iteration the new pattern is extended with another M-pattern of `J`. The third iteration shows an addition of the M-pattern of `A`, resulting in a non-connected pattern and, thus, fixed with the addition of an arc ($\langle C \rightarrow D \rangle$) which connects the two patterns. Finally, the last iteration in the example is made with an arc ($\langle C \rightarrow E \rangle$), producing a pattern formed by two simple patterns.

traces. Therefore, a pattern is removed of the expansion stage when it becomes infrequent, because it will never be contained again in a frequent pattern.

Fig. 8 shows an example with 4 iterations of the expansion of a minimal pattern, assuming that the expanded patterns are frequent. Although the example shows only one path of expansion, in each iteration the algorithm generates as many patterns as M-patterns and arcs are successfully added.

The pseudocode in Alg. 1 shows the main structure of the search made by WoMine. First, the frequent arcs of $A^{<}$ and the frequent M-patterns are initialized using the algorithm described in Section 5 to measure the frequency —M is the set with all M-patterns of the model. These M-patterns will be used to start the iterative stage, and to expand other patterns with them. Also, the final set is initialized with them because they are valid frequent patterns (Alg. 1:2-7).

Afterwards, the iterative part starts (Alg. 1:8). In this stage, an expansion of each of the current patterns is done, followed by a filtering of the frequent patterns. The expansion by adding frequent arcs of the $A^{<}$ set (Alg. 1:11) is done with the function `addFrequentArcs` (Alg. 1:38-46). The other expansion, the addition of M-patterns that are not in the current pattern (Alg. 1:12-15), is done with the function `addFrequentMPattern` (Alg. 1:22-28). If the joint pattern is unconnected, it generates as many patterns as possible by adding arcs that connect the unconnected parts with the function `addFrequentConnection` (Alg. 1:29-37). Once the expansion is completed, the obtained patterns are filtered to delete the infrequent ones (Alg. 1:17). Finally, once the iterative stage finishes, a simplification is made to delete the patterns which provide redundant information (Alg. 1:20). This simplification stage is explained in detail in Section 6.

WoMine is a robust algorithm, even for process models with low fitness, precision or generalization, as it extracts the patterns from the model, but measures the frequency with the log. If a structure is supported by the log, but it does not appear in the model (low fitness), it will not be considered as a frequent pattern. Anyway, this situation is irrelevant because, unless the model has a very low fitness, the unsupported structures will have low frequency. Moreover, the patterns detected by WoMine are not affected by models with high generalization —models that allow behaviour not recorded in the log—: the non-existent structures in the log have a frequency of 0 and, thus, will never be detected by WoMine.

5. Measuring the Frequency of a Pattern

In each step of the iterative process, WoMine reduces the search space by pruning the infrequent patterns (Alg. 1:17). For this, an algorithm to check the frequency of a pattern is needed (Alg. 2). Following Defs. 9 and 10, the algorithm generates the simple patterns of a pattern and checks the frequency of each one (Alg. 2:2-6). After calculating the frequency of the simple patterns, the function checks if this is considered frequent w.r.t. the thresh-

Algorithm 1. Main structure of WoMine.

```
Input: A process model $W$, a set $T = \{T_1, T_2, \ldots, T_n\}$ of traces of $W$, and a threshold $thr$.
Output: A set of maximum frequent patterns of $W$ w.r.t. $T$.
Algorithm WoMine (W, T, thr)
    $M \leftarrow \{m \mid m \in W, m$ is an $M$-pattern $\}$ // Def. 6
    $A^{<} \leftarrow \{a \mid a \in W, a$ is a Candidate Arc $\}$ // Def. 7
    $frequentArcs \leftarrow \{a \mid a \in A^{<}, a$ is frequent w.r.t. $T\}$
    $frequentM \leftarrow \{m \mid m \in M,$ isFrequentPattern($m, T, thr$) $\}$ // using Alg. 2
    $frequentPatterns \leftarrow frequentM$
    $currentPatterns \leftarrow frequentM$
    while $currentPatterns \neq \emptyset$ do
        $candidatePatterns \leftarrow \emptyset$
        forall $p \in currentPatterns$ do
            $candidatePatterns \leftarrow candidatePatterns \cup$ addFrequentArcs ($p$)
            $complementaryM \leftarrow \{m \mid m \in M, m \notin p\}$
            forall $m \in complementaryM$ do
                $candidatePatterns \leftarrow candidatePatterns \cup$ addFrequentMPattern ($p, m$)
            end
        end
        $currentPatterns \leftarrow \{p \mid p \in candidatePatterns,$ isFrequentPattern($p, T, thr$)$\}$ // using Alg. 2
        $frequentPatterns \leftarrow frequentPatterns \cup currentPatterns$
    end
    Delete the redundant patterns of $frequentPatterns$ // Section 6
    return $frequentPatterns$
Function addFrequentMPattern ($p, m$)
    $p' \leftarrow$ add $m$ to $p$
    if $p'$ is connected then
        return $p'$
    else
        return addFrequentConnection ($p', p, m$)
    end
Function addFrequentConnection ($p', p, m$)
    $patterns \leftarrow \emptyset$
    forall $arc \in frequentArcs$ do
        if ($arc.source \in p$ && $arc.destination \in m$) || ($arc.source \in m$ && $arc.destination \in p$) then
            $q \leftarrow$ add $arc$ to $p'$
            $patterns \leftarrow patterns \cup q$
        end
    end
    return $patterns$
Function addFrequentArcs ($p$)
    $patterns \leftarrow \emptyset$
    forall $arc \in freqArcs$ do
        if $arc \notin p$ && $arc.source \in p$ && $arc.destination \in p$ then
            $q \leftarrow$ add $arc$ to $p$
            $patterns \leftarrow patterns \cup q$
        end
    end
    return $patterns$
```

old and returns the corresponding value (Alg. 2:12). The frequency of a simple pattern is measured in the function `getPatternFrequency` by parsing all the traces and checking how many of them are compliant with it (Alg. 2:15-19). Finally, to check if a trace is compliant with a simple pattern, the function `isTraceCompliant` is executed: it goes over the activities in the trace (Alg. 2:22), simulating its execution in the model, and retrieving the activities that have fired the current one (Alg. 2:24-25). The simulation (`simulateExecutionInPattern`) consists in a replay of the trace, checking if the pattern is executed correctly.

With the current activity —the fired one— and the activities that have fired it —the firing activities, retrieved by the simulation—, the executed activities and arcs are saved, in order to analyse and to detect if the execution of the pattern is being disrupted before it is completed (Alg. 2:25). Fig. 9 shows an example of this process. The algorithm starts (#0) with the sets of the *executed arcs* and *last executed activities* empty. The first step (#1) executes `A`. There are no firing activities because `A` is the initial activity of the process model. As `A` is also one of the initial activities of the pattern, it is saved correctly in the *last executed activities* set.

The following activity (#2) in the trace is `B`. As there is only one firing activity (`A`), a single arc is executed ($\langle A \rightarrow B \rangle$). The arc is added to the *executed arcs* set, and the activity `B` to the *last executed activities* set. The `A` activity is not deleted because the set of outputs is formed by `{B, C}`, and `C` is still pending.

Algorithm 2. Check if a given pattern is executed more times than a threshold.

```
   Input: A set $T = T_1, ..., T_N$ of traces, a pattern pattern to measure its frequency w.r.t. $T$ and a threshold to establish the bound of
          frequency.
   Output: A Boolean value indicating if the pattern is frequent or not.
1  Algorithm isFrequentPattern (pattern, T, threshold)
2      simplePatterns ← generate the simple patterns of pattern
3      frequencies ← $\emptyset$
4      forall simplePattern ∈ simplePatterns do
5          frequencies ← frequencies ∪ getPatternFrequency (simplePattern, T)
6      end
7      minFreq ← 0
8      if frequencies.length > 0 then
9          minFreq ← minimum of frequencies
10     end
11     realFreq ← minFreq/T.length
12     return realFreq ≥ threshold
13 Function getPatternFrequency (pattern, T)
14     executed ← 0
15     forall trace ∈ T do
16         if isTraceCompliant (pattern, trace) then
17             executed ← executed + 1
18         end
19     end
20     return executed
21 Function isTraceCompliant (pattern, trace)
22     forall activity ∈ trace do
23         Execute activity in the process model
24         sources ← get the activities that fired the execution of activity
25         simulateExecutionInPattern (sources, activity, pattern)
26         if pattern has been successfully executed then
27             return true
28         end
29     end
30     return false
```

The next four steps, activities E (#3), G (#4), J (#5) and G (#6), will have the same behaviour. They have only one firing activity, i.e., one executed arc. The arcs are in the pattern and their source activities are in the *last executed activities* set, because they were executed before them. Hence, after adding and removing these activities from the last executed ones, G is the remaining one. After this process, the following activity is C (#7). Its execution has the same behaviour as the execution of B, but with the deletion of A from the *last executed activities*, because the set of outputs {B, C} has been fired.

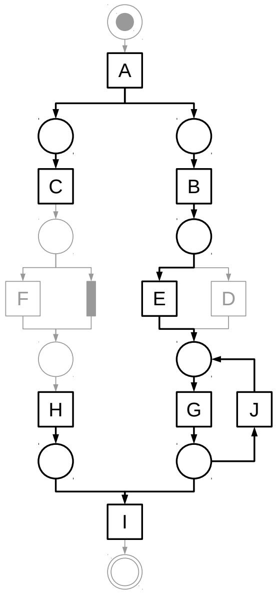
At step #8, only the source activity of the arc $\langle C \rightarrow F\rangle$ is in the pattern. In this case, the source of the arc is one of the end activities and, thus, the pattern finishes its execution in that branch. The execution is done with no action in the sets *executed arcs* and *last executed activities*.

In the next iteration (#9) only the target activity of the executed arc $\langle F \rightarrow H\rangle$ is in the pattern. As the target is one of the initial activities, the pattern starts to be executed in that branch. Thus, in a similar way to A, H is added to the *last executed activities* set.

Finally (#10), I has two firing activities and, thus, two arcs are executed. In both cases, the source activity of the arcs —G and H— is in the *last executed activities* set, and the arc is in the pattern. Thus, a simple addition of I to the *last executed activities* set is done when the last of its branches is executed.

At the end of each step, the algorithm checks if the pattern has been correctly executed (Alg. 2:26), i.e., all its arcs have been correctly executed and the *last executed activities* set corresponds with the end activities of the pattern. Unlike the other steps, this testing has a positive result when I is executed. Thus, the trace is compliant with the pattern.

The process of saving the executed arcs and activities has to be restarted when the executed arc is disrupting the execution of the pattern. For instance, in step #3, if the arc $\langle B \rightarrow D\rangle$ was executed instead of $\langle B \rightarrow E\rangle$, this would



(a) Petri net of a process model with a pattern highlighted in black (the unnamed activity is an invisible activity, i.e., an activity that is fired automatically to simulate the arc ⟨$C \rightarrow H$⟩).

Trace: `A B E G J G C F H I`			
Initial activities: `[A, H]` **End activities**: `[C, I]`			
#	**executed activity**	**executed arcs**	**last executed activities**
0	-	$\emptyset$	$\emptyset$
1	A	$\emptyset$	A
2	B	$\langle A \rightarrow B\rangle$	A, B
3	E	$\langle A \rightarrow B\rangle, \langle B \rightarrow E\rangle$	A, E
4	G	$\langle A \rightarrow B\rangle, \langle B \rightarrow E\rangle, \langle E \rightarrow G\rangle$	A, G
5	J	$\langle A \rightarrow B\rangle, \langle B \rightarrow E\rangle, \langle E \rightarrow G\rangle, \langle G \rightarrow J\rangle$	A, J
6	G	$\langle A \rightarrow B\rangle, \langle B \rightarrow E\rangle, \langle E \rightarrow G\rangle, \langle G \rightarrow J\rangle, \langle J \rightarrow G\rangle$	A, G
7	C	$\langle A \rightarrow B\rangle, \langle B \rightarrow E\rangle, \langle E \rightarrow G\rangle, \langle G \rightarrow J\rangle, \langle J \rightarrow G\rangle, \langle A \rightarrow C\rangle$	G, C
8	F	$\langle A \rightarrow B\rangle, \langle B \rightarrow E\rangle, \langle E \rightarrow G\rangle, \langle G \rightarrow J\rangle, \langle J \rightarrow G\rangle, \langle A \rightarrow C\rangle$	G, C
9	H	$\langle A \rightarrow B\rangle, \langle B \rightarrow E\rangle, \langle E \rightarrow G\rangle, \langle G \rightarrow J\rangle, \langle J \rightarrow G\rangle, \langle A \rightarrow C\rangle$	G, C, H
10	I	$\langle A \rightarrow B\rangle, \langle B \rightarrow E\rangle, \langle E \rightarrow G\rangle, \langle G \rightarrow J\rangle, \langle J \rightarrow G\rangle, \langle A \rightarrow C\rangle, \langle G \rightarrow I\rangle, \langle H \rightarrow I\rangle$	C, I

(b) Check of the execution of a trace for the pattern highlighted in Fig. 9a: '#' is the step of the algorithm; *'executed activity'* is the activity currently executed; *'executed arcs'* is the set with the arcs belonging to the pattern which execution was correctly saved; *'Last executed activities'* is the set of activities which have not fired an entire set of their outputs.

Figure 9: An example that shows how the algorithm checks if a trace is compliant with a pattern of the process model.

cause this saving process to go back by removing the arcs and activities of the failed path and to continue with the trace in order to check if the execution of the pattern is resumed later. When an arc outside the pattern —for example of a concurrent branch— is executed, the analysis is not disrupted because the execution does not correspond to the pattern. This analysis is able to recognize the correct execution of a pattern in 1-safe Petri nets[7].

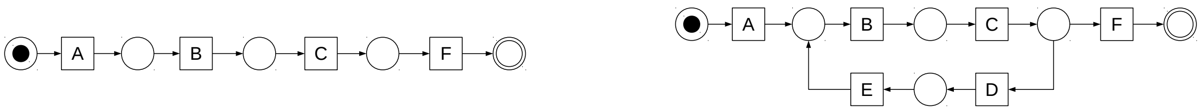
6. Simplifying the Result Set of Patterns

The result set of the a priori search has a high redundancy. This is because there are patterns in the k-th iteration which are expanded and thus are subpatterns of those in the $k + 1$-th iteration. A naive approach to reduce the redundancy generated by the expansion might be to remove the patterns from iteration k-th that are expanded in iteration $k + 1$-th. But, with the existence of loops, this naive approach might fail (see Fig. 10 for an example).

In Womine, the simplification process deletes the patterns contained into others, but only when the behaviour of the contained pattern is also included in the other pattern. For this, each pattern is compared with its previous patterns in the expansion. If all activities and arcs of a previous pattern are contained into the current one, and there is no new loops in the current pattern, the previous one is deleted. For this, an algorithm to detect loop arcs (Section 6.1) is executed for the current pattern. For instance, if both 10b and 10a were in the results set, WoMine would detect that 10a is inside 10b, but, as the difference between them begins in the start of a loop and finishes in the end, the pattern is not deleted. The next section explains the approach designed to identify the arcs starting and closing a loop.

6.1. Identification of the start and end arcs of a loop

[7] A Petri net is 1-safe when the value of the places can be binary, i.e., there can be only one mark in a place at the same time.



(a) Simple pattern formed by a sequence.

(b) Simple pattern with a 2-length loop.

Figure 10: An example where the naive simplification fails. With this naive technique, assuming a scenario where 10a and 10b were frequent patterns, 10a would be removed from the frequent results —because 10b would be obtained, among others, by an expansion of 10a. Thus, the apparition of 10b as a frequent pattern would mean that both behaviours, the sequence (*A-B-C-F*) and the pattern with the loop (*A-B-C-D-E-B-C-F*), are frequent. Therefore, with the naive technique, it is impossible to indicate that the pattern with the loop appears frequently in the traces, but the sequence does not. Because the apparition of pattern 10b in the results indicates both behaviours as frequent.

Definition 11 (Startloop arc). Given a process model composed by a set of activities A, and a set of arcs D, a *Startloop* arc $e \in D$ is an arc that starts a loop, i.e., e starts a path that will result, inevitably, in a return to a previous activity in the process.

Definition 12 (Endloop arc). Given a process model composed by a set of activities A, and a set of arcs D, an *Endloop* arc $e \in D$ is an arc that ends a loop, i.e., the transition through e implies the immediate closing of a loop and a return to a previous activity in the process.

An example of a Startloop arc is $\langle C \rightarrow D \rangle$ in Fig. 10b, while $\langle E \rightarrow B \rangle$ is an Endloop arc. Alg. 3 identifies the Startloop and Endloop arcs of a pattern. The approach is an iterative process with two different phases in each iteration. First, it searches the Startloop arcs (Alg. 3:4) and then, based on these arcs, it looks for the arcs that close the loops —Endloop arcs— (Alg. 3:5-12).

Startloop search

The search starts with the initial activities of the pattern and goes forward until it reaches as activity with more than one output. When this happens, an analysis trying to close a loop —reach the current activity again— is thrown for each output arc. The analysis goes forward through non-Startloop arcs and finishes when it reaches the current activity or the end of the pattern. If the analysis reaches the starting activity, the arc is marked as Startloop.

Table 2 shows an example of this search for the pattern of Fig. 10b. It starts at `A` (Alg. 3:18), and stops at `C` in step #3, because it has more than one output (Alg. 3:23). With its output arcs, $\langle C \rightarrow F \rangle$ and $\langle C \rightarrow D \rangle$, a subanalysis to detect if any of them is the beginning of a loop starts (Alg. 3:25). This analysis performs a depth-first search going forward through the arcs that are not detected as Startloop, trying to find the source of the arc that started the analysis, i.e., trying to close the loop. With $\langle C \rightarrow F \rangle$ it reaches the end of the model and stops the search (#4). But with $\langle C \rightarrow D \rangle$ it closes the loop reaching `C`, after going through `D` (#5), `E` (#6) and `B` (#7). In this case, the arc under analysis, $\langle C \rightarrow D \rangle$, is marked as Startloop (#8).

Endloop search

This search starts with the target activity of each Startloop detected in the same iteration. For each activity, it goes forward through its output arcs by analysing their target activities. The analysis continues until an activity that can reach, going backwards, the start of the pattern is found. When the start is reached, the current arc is marked as Endloop. In this backwards search, the algorithm cannot go through a Startloop arc detected in the same iteration.

Table 3 shows an example of this search for the pattern of Fig. 10b. As there is only one Startloop detected in this iteration ($\langle C \rightarrow D \rangle$), the search begins with it. The search tries to go backwards from `D` (the target of the Startloop arc) in step #1 (Alg. 3:7), but as the unique input arc is one of the detected Startloop arcs, it stops and goes forward to `E` in step #2 (Alg. 3:10). From `E` the same happens: it goes backwards to `D` (#3) and stops again. Finally, it searches from `B` (#4), where the algorithm is able to reach the initial activity going backwards through the $\langle A \rightarrow B \rangle$ arc (#7). Therefore, the current arc, $\langle E \rightarrow B \rangle$, is marked as Endloop.

The iterative nature of the algorithm allows it to find loops inside other loops, and to detect also multiple Startloop or Endloop arcs for the same loop, i.e., loops with more than one input or more than one output.

Algorithm 3. Identify the Startloop and Endloop arcs of a pattern or process model.

```
Input: A pattern p
Output: The pattern p with the Startloop and Endloop arcs identified
Algorithm searchLoopArcs(p)
    startloopArcs ← ∅
    do
        startloopArcs ← searchStartloopArcs(p, startloopArcs)
        forall arc ∈ startloopArcs do
            startActivity ← arc.target
            if arriveStartWithoutLoop(startActivity, startloopArcs) then
                set arc as Endloop
            else
                continueWithOutputs(startActivity, startloopArcs)
            end
        end
    while startloopArcs ≠ ∅
    return p
Function searchStartloopArcs(p, previousStartloops)
    initialActivities ← ∅
    if previousStartloops = ∅ then
        initialActivities ← start activities of p
    else
        initialActivities ← targets of arcs in previousStartloops
    end
    forall activity ∈ initialActivities do
        Go forward through the outputs while there is only one
        forall output ∈ sorted activity.outputs do
            analizeSplit(activity, output)
        end
    end
    return the set of new Startloops
Function continueWithOutputs(previousActivity, startloopArcs)
    forall output ∈ previousActivity.outputs do
        if arriveStartWithoutLoop(output, startloopArcs) then
            set ⟨previousActivity → output⟩ as Endloop
        else
            if output has not been explored then
                continueWithOutputs(output, startloopArcs)
            end
        end
    end
```

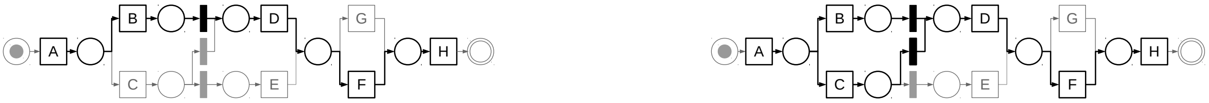
#	activity under analysis	outputs	subanalysis			action
			possible Startloop arc	**current activity**	**outputs**	
1	A	B	-	-	-	only one output, continue
2	B	C	-	-	-	only one output, continue
3	C	F, D	-	-	-	two outputs, start subanalysis searching for C
4			$\langle C \rightarrow F \rangle$	F	$\emptyset$	reached end of model, C not found
5			$\langle C \rightarrow D \rangle$	D	E	C not found, continue with outputs
6				E	B	C not found, continue with outputs
7				B	C	C found, set $\langle C \rightarrow D \rangle$ as Startloop
8	-	-	-	-	-	No more activities, end of first phase

Table 2: Startloop search for the pattern of Fig. 10b: *'activity under analysis'* is the activity which output arcs are currently examined; *'outputs'* are the outputs of the activity under analysis, i.e., the targets of the arcs; *'possible Startloop arc'* is the arc being analysed; *'current activity'* is the current activity in the search of the source activity of the arc; *'outputs'* is the set of outputs of the 'current activity', i.e., the next in the analysis; *'action'* is a description of the action at the end of each iteration.

7. Experimentation

The validation of the presented approach has been done with different types of event logs. Subsection 7.1 presents the results of the comparison between WoMine and the state of the art techniques for 5 process models. Subsection 7.2 discusses, for 20 logs from [6], the extracted frequent patterns and their evolution as the threshold varies. Finally, in Subsection 7.3, we prove the performance of WoMine over complex real logs and compare the impact of the model quality in the extraction of patterns using several Business Process Intelligence Challenge's logs.

These experiments have been executed in a laptop (Lenovo G500) with an Intel i7-3612QM (2.1 GHz) processor



(a) WoMine pattern (highlighted) for a threshold of 40%.

(b) Heat maps pattern (highlighted) for a threshold of 40. The pattern includes infrequent paths, for instance `A-C-D-H-F`. These arcs are frequent individually, but not the sequence itself. This happens because the executions of $\langle A \rightarrow C\rangle$ are distributed in traces through `D-G` and `E`.

Figure 11: Results of WoMine and heat maps for a process model with several selections.

and 8GB of RAM (1600 MHz). Although WoMine has been described for Causal nets, the algorithm[8] is able to mine patterns both from models represented with Causal net [27] and with Causal matrix [28] (Heuristics net).

7.1. Comparison between WoMine and the state of the art approaches

In this comparison, 5 process models with the most common control structures have been used. These models present scenarios where WoMine is able to retrieve frequent patterns, while any of the other techniques fails. For each model, two Petri nets will be presented: one with a highlighted frequent pattern extracted by WoMine, and another one with the frequent structure extracted by heat maps. The structure obtained by heat maps is retrieved establishing a threshold, and highlighting all the arcs with a frequency over it. The chosen threshold is the one that allows to get the structure closest to the frequent pattern extracted by WoMine. Table 4 shows the results of these techniques for the 5 process models.

The first process model (Fig. 11) has several selections. WoMine finds a pattern appearing in the 40% of the traces (Fig. 11a). On the contrary, the heat maps discovers, as frequent, paths that are not frequent. The other approaches — *w*-find, local process mining, episode mining, sequential pattern mining (PrefixSpan), and the tree mining approach— detect the same pattern as WoMine.

Fig. 12 presents the second process model, which has a loop. WoMine finds a frequent pattern appearing in the 70% of the traces (Fig. 12a). The heat maps approach, nevertheless, retrieves as frequent the structure with the execution of the loop (Fig. 12b). *w*-find gets the same pattern as WoMine, but with a wrong frequency. As has been explained before, when `H` is executed in a trace, *w*-find can not distinguish if it is a loop disrupting the execution of the pattern, or an activity in other parallel branch of the model. In the local process mining technique, the pattern is also registered as executed in the traces with `H`, retrieving the same result as *w*-find. The episode mining approach also retrieves, among other patterns with the same activities but different relations, this pattern with a wrong frequency

[8] The algorithm can be tested and downloaded from `http://tec.citius.usc.es/processmining/womine/`

#	Startloop	possible Endloop	activity to reach the start from	pending to analyse	input under analysis	action
1	$\langle C \rightarrow D\rangle$	$\langle C \rightarrow D\rangle$	D	C	C	$\langle C \rightarrow D\rangle \in$ detected Startloop arcs, cannot go back through this path
2		$\langle D \rightarrow E\rangle$	E	D	D	keep going back
3			D	C	C	$\langle C \rightarrow D\rangle \in$ detected Startloop arcs, cannot go back through this path
4		$\langle E \rightarrow B\rangle$	B	A, E	E	keep going back
5			E	A, D	D	keep going back
6			D	A, C	C	$\langle C \rightarrow D\rangle \in$ detected Startloop arcs, cannot go back through this path
7			A	-	-	Start of model reached, $\langle E \rightarrow B\rangle$ marked as Endloop arc

Table 3: Endloop search for the pattern of Fig. 10b: *'startloop'* is the Startloop, of the detected ones in this iteration, to start the search; *'possible Endloop'* is the arc considered as Endloop if the start of the pattern is reached from its target activity; *'activity to reach the start from'* is the activity from which the search is trying to reach the start in that step; *'pending to analyse'* is the set of input activities that have not been analysed in the search for the start of the pattern; *'input under analysis'* is the activity currently being analysed in the search for the start of the pattern; *'action'* is a description of the action at the end of each iteration.

	Examples				
	#1 (Fig. 11)	**#2 (Fig. 12)**	**#3 (Fig. 13)**	**#4 (Fig. 14)**	**#5 (Fig. 15)**
WoMine	**+**	**+**	**+**	**+**	**+**
Heat Maps [14]	±	-	-	+	-
w-find [11]	+	±	-	-	-
Local Process Mining [24]	+	±	±	-	±
Episode Mining [19]	+	±	-	-	-
SPM (PrefixSpan) [21]	+	-	±	-	-
Tree Mining [3]	+	-	±	-	-

Table 4: Comparison between WoMine and other state of the art techniques for 5 process models: '+' stands for a correct frequent pattern extraction; '-' stands for a non extraction of the frequent pattern, and '±' stands for an incorrect extraction of the frequent pattern (similar but wrong structure or wrong frequency).

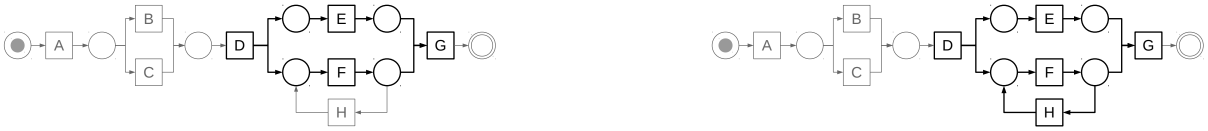
due to the same reason. PrefixSpan and the tree mining approach cannot get structures with parallels because they interpret that traces with E before F are different than those with F before E.

In the third scenario, Fig. 13a depicts interesting behaviour found by WoMine. In the 55% of the traces, the pattern performs the sequence through D, followed by the loop with J and D again, without including neither I nor E. On the contrary, the heat maps (Fig. 13b) highlights wrongly the pattern with the two loops as frequent. The *w*-find approach cannot retrieve the pattern found by WoMine due to the existence of a loop. With the local process mining approach, the pattern is found, but the frequency is incorrect when the loop (J) is not executed, or when I is executed, because the final mark is reached in both cases. The episode mining approach cannot find this pattern in a feasible time, due to the number of combinations among activities that it checks. PrefixSpan gets the pattern retrieved by WoMine replacing the loop with a sequence with one repetition, i.e., duplicating the activities in the sequence. Also, PrefixSpan does not ensure that I or E are not executed. The tree mining approach presents the same drawback, but in a tree structure.

WoMine finds, for the process model in Fig. 14, two frequent patterns composed each one by an arc (Fig. 14a) and separated by a selection. The knowledge extracted by heat maps (Fig. 14b) agrees with the result of WoMine. All the other techniques retrieve, among others, the sequence A−B−G−H as frequent with a 65%. This is because they do not take into account the process model while checking the frequency of the pattern, and they consider that the execution of F is due to the other parallel branch in the model.

Finally, in the last case (Fig. 15), Fig. 15a shows a frequent structure executed in the 55% of the traces. The structural complexity of this pattern makes impossible its extraction by the heat maps approach (Fig. 15b). Similar to the Fig. 13 case, *w*-find cannot retrieve the pattern, local process mining retrieves it with a wrong frequency, and the episode mining approach fails due to the high number of possible relations between the activities. PrefixSpan and the tree mining approach cannot detect this pattern due to the parallel structures.

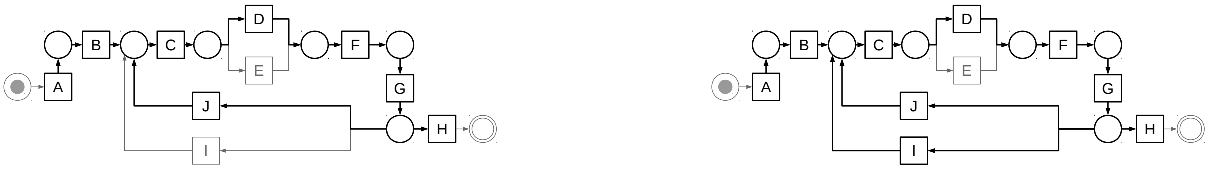
In summary, the state of the art algorithms fail to detect several patterns that can be retrieved with WoMine. The heat maps approach is very simple, and allows to extract interesting behaviour in a quick way, but with the inability to detect which activity causes the firing of another activity. The *w*-find algorithm uses the model to extract frequent patterns but, the approach to check the compliance of a trace with a pattern fails in some cases. Loops and choices are



(a) WoMine pattern (highlighted) for a threshold of 70%. The traces compliant to this pattern are those where the loop is not executed.

(b) Heat maps pattern (highlighted) for a threshold of 70. The parallel structure with the loop is not frequent. The loop is executed several times in the same trace, increasing its absolute frequency, but not the frequency of the whole pattern.

Figure 12: Results of WoMine and heat maps for a process model with a 1-length loop inside a branch with a parallel structure.



(a) WoMine pattern (highlighted) for a threshold of 55%. This structure corresponds the instances of the process model where `E` and `I` are not executed, and the loop of `J` is executed at least one time.

(b) Heat maps pattern (highlighted) for a threshold of 100. The arcs of the `I` loop are individually frequent, but the highlighted structure is not.

Figure 13: Results of WoMine and heat maps for a process model composed by a sequence with a selection, and two loops sharing the start and end activities. The loops give the model the ability to execute both branches of the selection in the same trace, and more than once.

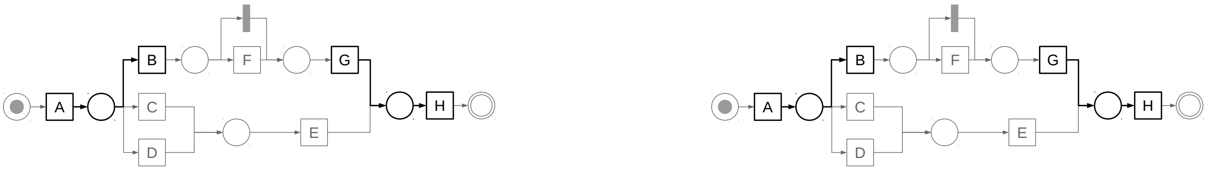
also unsupported by this algorithm. The local process mining technique does not ensure that the pattern was entirely executed, without disruptions in the middle of its execution. Moreover, it does not ensure that the loops of a pattern were executed when the final marking is reached either. The episode mining approach does not use the process model, which causes the extraction of a high number of similar patterns varying the relations between the activities. Also, two sequences separated by infrequent selections are detected as one single sequence by this technique. The sequential pattern mining search, represented by PrefixSpan, has a similar problem. Moreover, it can only detect sequences. Finally, the tree mining approach obtains results similar to sequential pattern mining, but searching in a tree structure.

7.2. Characteristics of the patterns and analysis of runtimes

This subsection presents the results obtained by WoMine in a set of process models for different thresholds. Tables 5 and 6 show the result of WoMine for 20 process models and three different thresholds. For instance, for process model *g25*, with a threshold of the 40%, WoMine discovered four patterns. These patterns have a frequency close to the 50% and, in average, 6.5 activities per pattern, with a standard deviation of 4.43, containing all kind of structures —sequences, choices, parallels and loops. As explained in Sec. 5, the algorithm needs to execute the trace in the model to retrieve the executed arcs. This process is independent of the threshold —it only depends on the traces (log) and on the model. Thus, the runtime is divided in two parts to distinguish this preprocessing time and the time spent by the algorithm.

The event logs were randomly generated —up to 300 traces— from process models with different complexity levels, ranging from 20 to 30 unique activities, and containing loops, parallelisms, selections, etc. A more detailed description of the behavioral structures of these process models can be found in [6, 34]. Furthermore, as WoMine takes as starting point a process model and an event log, we used ProDiGen [34] over this set of event logs to retrieve the process model.

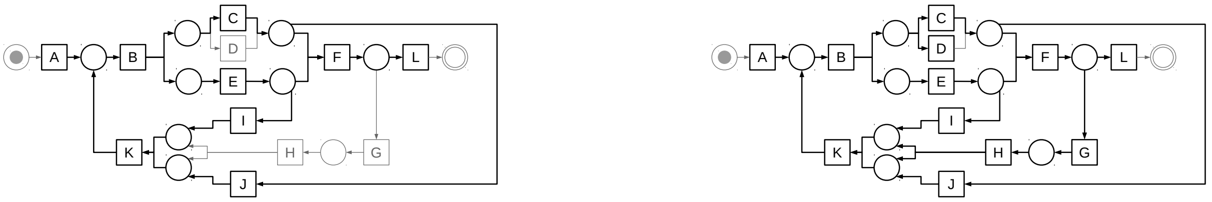
As can be seen, WoMine is able to retrieve frequent patterns with all type of structures. When the threshold increases, WoMine obtains less and simpler patterns. This is because, as the minimum frequency is increased, more patterns become infrequent and stop belonging to the result set, which also reduces the possibilities for growing the



(a) WoMine pattern (highlighted) for a threshold of 65%.

(b) Results of the heat maps approach applied with a threshold of 65. The result is correct and agrees with WoMine's.

Figure 14: Results of WoMine and heat maps for a process model with a selection of three branches, having one of them an optional activity.



(a) WoMine pattern (highlighted) for a threshold of 55%. The pattern consists in a parallel structure with `C` in the upper branch, the loop of `I-J`, and passing again through `C`.

(b) Heat maps pattern (highlighted) for a threshold of 55. The repetition of the loops increases the frequency of other arcs, without building a recognizable pattern.

Figure 15: Results of WoMine and heat maps for a process model with two loops sharing the end activity, and one of them starting in a parallel structure.

patterns. Nevertheless, there are some cases where the number of frequent patterns becomes higher when the threshold increases (g10, g13, g21, etc.). This happens when a large pattern is splitted due to the increase of the threshold, as some parts of it become infrequent and trigger a disjointed structure. For instance, model *g10* has a frequent pattern with 14 activities for a threshold of 40%, and two patterns for 60% but with an average of 2.5 activities per pattern.

Regarding the runtime of the algorithm, the preprocessing time is always under 60 ms, usually 20 ms. This is the time to parse the 300 traces, and retrieve the executed arcs. On the other hand, the runtime of the algorithm decreases when the threshold increases, as more patterns become infrequent and are pruned earlier, saving computational cost. The global runtime —preprocessing plus algorithm— is under 500 milliseconds in all cases except the executions of *g3* and *g7*, both with thresholds of 40% and 60%.

7.3. Frequent patterns for the BPI Challenges

The objective of this subsection is twofold: on the one hand, to test WoMine on complex real logs from the Business Process Intelligence Challence (BPIC) [29] and, on the other hand, to analyze the influence of the model in the retrieved patterns. Due to the complexity of the models mined for some BPIC logs, in these experiments the process models are represented through the Causal matrix formalism.

	Threshold : 40%								
	runtime (secs)		#patt	frequency	#activities	#sequences	#choices	#parallels	#loops
	pre	alg							
g2	0.005	0.011	2	0.48±0.04	4.50±0.71	1.00±0.00	0±0	0±0	0±0
g3	0.060	6.800	4	0.45±0.04	14.25±5.12	1.75±0.96	1.50±1.29	3.00±0.82	0.25±0.50
g4	0.010	0.236	4	0.62±0.25	4.50±2.38	0.75±0.50	0.25±0.50	0±0	0.25±0.50
g5	0.006	0.066	3	0.50±0.03	8.00±5.29	1.00±1.00	0.67±0.58	1.33±1.15	0±0
g6	0.007	0.084	3	0.67±0.29	6.67±3.06	0.67±0.58	0.67±1.15	0.67±1.15	0±0
g7	0.046	6.029	8	0.48±0.04	16.00±2.20	1.75±0.46	1.00±1.20	1.75±0.71	0.25±0.46
g8	0.015	0.082	4	0.72±0.33	4.25±1.26	0.75±0.50	0.25±0.50	0.50±1.00	0±0
g9	0.007	0.039	3	0.50±0.02	6.33±0.58	1.00±0.00	0.33±0.58	0.33±0.58	0±0
g10	0.006	0.043	1	0.49±0.00	14.00±0.00	1.00±0.00	2.00±0.00	2.00±0.00	0±0
g12	0.009	0.061	3	0.67±0.29	6.00±3.00	1.00±0.00	0.33±0.58	0.67±1.15	0±0
g13	0.002	0.025	1	0.48±0.00	13.00±0.00	2.00±0.00	2.00±0.00	0±0	0±0
g14	0.019	0.179	5	0.59±0.23	6.00±1.87	1.20±0.45	0±0	0±0	0.60±0.55
g15	0.013	0.009	3	0.76±0.24	2.67±1.15	0.33±0.58	0±0	0±0	0±0
g19	0.002	0.015	1	0.47±0.00	11.00±0.00	2.00±0.00	1.00±0.00	2.00±0.00	0±0
g20	0.022	0.235	6	0.58±0.21	4.67±2.66	0.83±0.75	0.50±0.55	0±0	0±0
g21	0.001	0.007	2	0.75±0.36	5.00±4.24	0.50±0.71	0.50±0.71	0±0	0±0
g22	0.001	0.006	1	0.44±0.00	8.00±0.00	1.00±0.00	1.00±0.00	1.00±0.00	0±0
g23	0.052	0.424	6	0.82±0.28	3.00±0.89	0.33±0.52	0±0	0±0	0.33±0.52
g24	0.002	0.014	4	0.65±0.24	3.50±1.91	0±0	0.25±0.50	0.75±0.96	0±0
g25	0.041	0.326	4	0.49±0.04	6.50±4.43	0.50±0.58	0.50±0.58	1.25±2.50	0.25±0.50

Table 5: Behavioral structure of the frequent patterns extracted, for a threshold of 40%, from the process models in [6]. The values show the runtimes for the preprocessing and for the algorithm; the number of patterns retrieved (*#patt*); the average and standard deviation of the frequency (*frequency*), number of activities (*#activities*), number of sequences (*#sequences*), number of choices (*#choices*), number of parallels (*#parallels*) and number of loops (*#loops*) per extracted pattern.

	Threshold : 60%								
	runtime (secs)		#patt	frequency	#activities	#sequences	#choices	#parallels	#loops
	pre	alg							
g2	0.005	0.006	2	0.87±0.18	3.50±0.71	1.00±0.00	0±0	0±0	0±0
g3	0.060	0.609	4	0.83±0.20	6.75±4.11	1.00±0.00	0.50±1.00	1.25±1.50	0.25±0.50
g4	0.010	0.011	4	1.00±0.00	3.25±1.89	0.50±0.58	0±0	0±0	0±0
g5	0.008	0.012	3	0.89±0.19	4.33±0.58	0.67±0.58	0±0	0.67±1.15	0±0
g6	0.007	0.014	2	1.00±0.00	4.00±2.83	0.50±0.71	0±0	0±0	0±0
g7	0.046	1.228	6	0.83±0.18	10.50±4.85	1.17±0.41	0.33±0.82	1.33±1.03	0.17±0.41
g8	0.015	0.010	2	1.00±0.00	3.50±0.71	1.00±0.00	0±0	0±0	0±0
g9	0.007	0.011	2	1.00±0.00	4.00±1.41	1.00±0.00	0±0	0±0	0±0
g10	0.006	0.004	2	1.00±0.00	2.50±0.71	0.50±0.71	0±0	0±0	0±0
g12	0.009	0.002	1	1.00±0.00	3.00±0.00	1.00±0.00	0±0	0±0	0±0
g13	0.002	0.006	2	1.00±0.00	5.50±0.71	1.00±0.00	0±0	0±0	0±0
g14	0.019	0.156	5	1.00±0.00	5.20±1.10	1.00±0.00	0±0	0±0	0±0
g15	0.013	0.005	3	0.92±0.14	2.33±0.58	0.33±0.58	0±0	0±0	0±0
g19	0.002	0.011	2	0.86±0.19	5.00±1.41	1.00±0.00	0±0	1.00±1.41	0±0
g20	0.022	0.014	3	1.00±0.00	2.67±0.58	0.67±0.58	0±0	0±0	0±0
g21	0.001	0.003	3	0.89±0.18	3.00±1.00	0.67±0.58	0±0	0±0	0±0
g22	0.001	0.001	2	1.00±0.00	2.50±0.71	0.50±0.71	0±0	0±0	0±0
g23	0.052	0.317	6	1.00±0.00	2.67±1.03	0.33±0.52	0±0	0±0	0±0
g24	0.002	0.001	2	1.00±0.00	2.00±0.00	0±0	0±0	0±0	0±0
g25	0.041	0.065	5	1.00±0.00	3.20±1.10	0.20±0.45	0±0	0.80±1.10	0±0
	Threshold : 80%								
	runtime (secs)		#patt	frequency	#activities	#sequences	#choices	#parallels	#loops
	pre	alg							
g2	0.005	0.004	2	1.00±0.00	3.00±1.41	0.50±0.71	0±0	0±0	0±0
g3	0.060	0.073	3	1.00±0.00	5.00±2.65	1.00±0.00	0±0	0.67±1.15	0±0
g4	0.010	0.010	4	1.00±0.00	3.25±1.89	0.50±0.58	0±0	0±0	0±0
g5	0.008	0.007	2	1.00±0.00	4.00±0.00	1.00±0.00	0±0	0±0	0±0
g6	0.007	0.008	2	1.00±0.00	4.00±2.83	0.50±0.71	0±0	0±0	0±0
g7	0.046	0.364	3	1.00±0.00	9.00±7.21	1.33±0.58	0±0	0.67±1.15	0±0
g8	0.015	0.009	2	1.00±0.00	3.50±0.71	1.00±0.00	0±0	0±0	0±0
g9	0.007	0.010	2	1.00±0.00	4.00±1.41	1.00±0.00	0±0	0±0	0±0
g10	0.006	0.004	2	1.00±0.00	2.50±0.71	0.50±0.71	0±0	0±0	0±0
g12	0.009	0.001	1	1.00±0.00	3.00±0.00	1.00±0.00	0±0	0±0	0±0
g13	0.002	0.006	2	1.00±0.00	5.50±0.71	1.00±0.00	0±0	0±0	0±0
g14	0.019	0.151	5	1.00±0.00	5.20±1.10	1.00±0.00	0±0	0±0	0±0
g15	0.013	0.004	2	1.00±0.00	2.50±0.71	0.50±0.71	0±0	0±0	0±0
g19	0.002	0.005	2	1.00±0.00	4.50±2.12	1.00±0.00	0±0	1.00±1.41	0±0
g20	0.022	0.012	3	1.00±0.00	2.67±0.58	0.67±0.58	0±0	0±0	0±0
g21	0.001	0.001	2	1.00±0.00	3.00±1.41	0.50±0.71	0±0	0±0	0±0
g22	0.001	0.001	2	1.00±0.00	2.50±0.71	0.50±0.71	0±0	0±0	0±0
g23	0.052	0.329	6	1.00±0.00	2.67±1.03	0.33±0.52	0±0	0±0	0±0
g24	0.002	0.001	2	1.00±0.00	2.00±0.00	0±0	0±0	0±0	0±0
g25	0.041	0.061	5	1.00±0.00	3.20±1.10	0.20±0.45	0±0	0.80±1.10	0±0

Table 6: Continuation of results in Table 5 for thresholds of 60% and 80%.

Table 7a shows some statistics of the BPIC logs [23, 29, 30, 31]. These logs have been mined with two of the most popular discovery algorithms, the Heuristics Miner (HM) [35] and the Inductive Miner (IM) [18]. Table 7b presents the characteristics of the mined models, which have been generated using ProM [32]. As can be seen, the models mined by IM contain many more arcs than the HM models. Also, models from years 2011 and 2015 are far more complex than models from other years.

A series of experiments have been run for these logs and models with different thresholds. Tables 8, 9 and 10 show the results for thresholds of 20%, 35% and 50%. We do not show higher thresholds because, for such complex models, the execution of a path many times is very uncommon. This would return a low number of patterns, with few structures, not allowing to study the differences between models. The results demonstrate the ability of WoMine to extract patterns with loops, choices, parallels and sequences.

		#traces	#events	events per trace		
				min	max	$\bar{X} \pm \sigma$
2011		1143	152577	3	1816	133.5±202.6
2012	fin	13087	288374	5	177	22.0±19.9
	a	4085	21565	5	10	5.3±0.8
	o	4038	32384	5	32	8.0±2.8
2013	inc	7554	80641	3	125	10.7±7.6
	clo	1487	9634	3	37	6.5±3.2
	op	819	3989	3	24	4.9±2.1
2015	1	1199	54615	4	103	45.6±17.0
	2	832	46018	3	134	55.3±19.9
	3	1409	62499	5	126	44.4±16.1
	4	1053	49399	3	118	46.9±15.0
	5	1156	61395	7	156	53.1±16.0

(a) Statistics of the BPIC logs. The BPIC 2012 logs *'a'* and *'o'* have been generated after a filtering in the BPIC 2012 raw log (*'fin'*), maintaining the traces which contain activities of the categories A and O, respectively.

		Heuristics Miner		Inductive Miner	
		#activities	#arcs	#activities	#arcs
2011		623	1480	626	390614
2012	fin	38	112	38	1044
	a	12	14	12	19
	o	9	17	9	28
2013	inc	15	101	15	171
	clo	9	34	9	28
	op	7	30	7	16
2015	1	400	719	400	153677
	2	412	747	412	162797
	3	383	697	385	142887
	4	358	635	358	113266
	5	391	733	391	147102

(b) Number of activities and arcs of the mined models, generated with two discovery algorithms: Heuristics Miner and Inductive Miner.

Table 7: Statistics about the logs of the BPICs and the mined model.

		Threshold : 20%																	
		Heuristics Miner									Inductive Miner								
		runtime (secs)		#patt	frequency	#activities	#sequences	#choices	#parallels	#loops	runtime (secs)		#patt	frequency	#activities	#sequences	#choices	#parallels	#loops
		pre	alg								pre	alg							
2011		11.894	114.632	21	0.25±0.08	5.57±4.02	0.62±0.67	0.81±1.08	0.29±0.46	0.43±0.51	-	-	-	-	-	-	-	-	-
2012	fin	8.434	12.838	7	0.29±0.05	3.14±2.91	0.29±0.49	0.14±0.38	0±0	0.29±0.49	17.339	100.683	6	0.25±0.07	6.50±4.46	0.33±0.52	0.83±1.17	0.83±2.04	0.67±0.52
	a	2.213	0.902	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0	1.581	0.776	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0
	o	0.962	2.791	2	0.24±0.01	5.50±0.71	0±0	1.00±1.41	1.50±0.71	1.00±1.41	1.096	0.186	2	0.75±0.35	2.00±0.00	0±0	0±0	0±0	0±0
2013	inc	2.810	123.813	6	0.25±0.06	5.33±2.25	0±0	1.17±0.98	0±0	0.67±0.82	2.893	123.432	6	0.25±0.06	5.33±2.25	0±0	1.17±0.98	0±0	0.67±0.82
	clo	0.560	3.941	4	0.24±0.01	4.50±1.29	0±0	0.75±0.50	0±0	0.25±0.50	0.366	0.081	2	0.76±0.34	1.50±0.71	0±0	0±0	0±0	0.50±0.71
	op	0.159	0.310	2	0.21±0.00	4.00±0.00	0.50±0.71	1.00±1.41	0±0	0±0	0.138	0.056	1	0.30±0.00	1.00±0.00	0±0	0±0	0±0	1.00±0.00
2015	1	2.821	0.910	14	0.32±0.12	2.57±0.76	0.21±0.43	0±0	0.21±0.43	0±0	41.419	95.367	19	0.24±0.05	4.79±2.74	0.32±0.48	1.05±1.18	0.11±0.32	0±0
	2	2.185	1.181	15	0.28±0.14	3.27±1.58	0.27±0.46	0.07±0.26	0.47±0.64	0±0	39.483	51.523	26	0.24±0.04	3.27±2.07	0.31±0.47	0.42±0.76	0±0	0±0
	3	3.714	1.628	14	0.31±0.11	2.50±0.94	0.07±0.27	0.07±0.27	0.14±0.36	0±0	50.466	230.615	30	0.23±0.04	4.87±2.47	0.37±0.49	1.30±1.29	0.10±0.40	0±0
	4	2.334	1.738	13	0.34±0.19	3.77±2.35	0.15±0.38	0.08±0.28	0.46±0.66	0±0	35.823	6.580	20	0.24±0.03	4.15±2.91	0.55±0.60	0.45±0.83	0±0	0±0
	5	3.056	2.092	9	0.27±0.07	4.56±1.81	0.44±0.53	0.22±0.44	0.56±0.53	0±0	50.637	6.773	21	0.24±0.03	3.86±2.39	0.43±0.51	0.48±0.75	0±0	0±0

Table 8: Behavioral structure of the frequent patterns extracted with a threshold of 20% from the process models of the BPICs. It shows the information for the results with two process models of each log (Heuristics and Inductive). The information contains the runtime, the number of patterns and the distribution (average and standard deviation) of the frequency, the number of activities, sequences, choices, parallels and loops of each pattern. The missing results in the 2011 log with the IM's model are due to a non convergence of the algorithm, taking more than 5 hours to execute a few iterations.

		Threshold : 35%																	
		Heuristics Miner									Inductive Miner								
		runtime (secs)		#patt	frequency	#activities	#sequences	#choices	#parallels	#loops	runtime (secs)		#patt	frequency	#activities	#sequences	#choices	#parallels	#loops
		pre	alg								pre	alg							
2011		11.506	3.395	14	0.44±0.07	2.64±1.45	0.36±0.50	0.14±0.36	0.07±0.27	0.36±0.50	-	-	-	-	-	-	-	-	-
2012	fin	8.192	2.864	4	0.38±0.01	4.00±2.45	0.50±0.58	0.25±0.50	0.25±0.50	0±0	17.076	5.330	4	0.38±0.01	4.25±2.87	0.50±1.00	0.25±0.50	1.25±2.50	0±0
	a	0.621	0.224	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0	0.621	0.232	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0
	o	0.901	0.397	2	0.75±0.35	3.50±2.12	0.50±0.71	0±0	0±0	0±0	1.060	0.187	2	0.75±0.35	2.00±0.00	0±0	0±0	0±0	0±0
2013	inc	2.757	4.005	2	0.43±0.09	4.00±1.41	0.50±0.71	0.50±0.71	0±0	1.00±1.41	2.910	3.936	2	0.43±0.09	4.00±1.41	0.50±0.71	0.50±0.71	0±0	1.00±1.41
	clo	0.308	0.123	2	0.87±0.10	2.50±0.71	0.50±0.71	0±0	0±0	0±0	0.319	0.043	2	0.76±0.34	1.50±0.71	0±0	0±0	0±0	0.50±0.71
	op	0.120	0.039	2	0.62±0.38	2.50±0.71	0.50±0.71	0±0	0±0	0±0	0.127	0.003	0	0±0	0±0	0±0	0±0	0±0	0±0
2015	1	2.846	0.398	7	0.49±0.10	2.43±0.79	0.14±0.38	0±0	0.14±0.38	0±0	41.775	0.544	7	0.49±0.10	2.57±1.13	0.14±0.38	0±0	0.14±0.38	0±0
	2	2.150	0.506	6	0.49±0.14	3.67±1.86	0.17±0.41	0±0	0.67±0.52	0±0	39.264	0.602	8	0.40±0.06	2.75±1.04	0.50±0.53	0.12±0.35	0±0	0±0
	3	3.192	0.520	6	0.44±0.10	2.67±0.82	0.33±0.52	0±0	0.17±0.41	0±0	45.592	0.706	11	0.47±0.11	2.27±0.65	0.09±0.30	0±0	0.09±0.30	0±0
	4	2.356	0.616	7	0.53±0.19	2.57±1.51	0±0	0±0	0.14±0.38	0±0	35.379	0.751	8	0.40±0.06	3.50±1.41	0.62±0.52	0±0	0±0	0±0
	5	2.923	0.875	6	0.44±0.15	4.33±1.63	0.33±0.52	0±0	0.83±0.41	0±0	50.728	0.737	8	0.38±0.03	3.00±1.41	0.38±0.52	0±0	0±0	0±0

Table 9: Behavioral structure of the frequent patterns extracted with a threshold of 35% from the process models of the BPICs. It shows the information for the results with two process models of each log (Heuristics and Inductive). The information contains the runtime, the number of patterns and the distribution (average and standard deviation) of the frequency, the number of activities, sequences, choices, parallels and loops of each pattern. The missing results in the 2011 log with the IM's model are due to a non convergence of the algorithm, taking more than 5 hours to execute a few iterations.

		Threshold : 50%																	
		Heuristics Miner									Inductive Miner								
		runtime (secs)		#patt	frequency	#activities	#sequences	#choices	#parallels	#loops	runtime (secs)		#patt	frequency	#activities	#sequences	#choices	#parallels	#loops
		pre	alg								pre	alg							
2011		11.072	1.101	9	0.53±0.03	2.44±1.01	0.33±0.50	0±0	0±0	0.22±0.44	-	-	-	-	-	-	-	-	-
2012	fin	7.944	0.761	2	0.78±0.31	2.50±0.71	0.50±0.71	0±0	0±0	0±0	17.258	0.708	2	0.78±0.31	2.50±0.71	0.50±0.71	0±0	0±0	0±0
	a	0.621	0.224	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0	0.620	0.223	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0
	o	0.890	0.416	2	0.75±0.35	3.50±2.12	0.50±0.71	0±0	0±0	0±0	1.060	0.103	2	0.75±0.35	2.00±0.00	0±0	0±0	0±0	0±0
2013	inc	2.779	2.386	4	0.67±0.15	2.75±0.50	0.25±0.50	0.50±0.58	0±0	0.25±0.50	2.883	2.470	4	0.67±0.15	2.75±0.50	0.25±0.50	0.50±0.58	0±0	0.25±0.50
	clo	0.309	0.125	2	0.87±0.10	2.50±0.71	0.50±0.71	0±0	0±0	0±0	0.325	0.041	2	0.76±0.34	1.50±0.71	0±0	0±0	0±0	0.50±0.71
	op	0.120	0.012	1	0.89±0.00	2.00±0.00	0±0	0±0	0±0	0±0	0.127	0.003	0	0±0	0±0	0±0	0±0	0±0	0±0
2015	1	2.902	0.195	3	0.63±0.03	2.67±0.58	0.33±0.58	0±0	0.33±0.58	0±0	41.596	0.410	4	0.61±0.06	2.75±1.50	0±0	0±0	0.25±0.50	0±0
	2	2.137	0.408	5	0.61±0.09	3.80±1.10	0±0	0±0	1.00±0.00	0±0	39.035	0.316	5	0.58±0.08	2.20±0.45	0.20±0.45	0±0	0±0	0±0
	3	3.189	0.280	3	0.68±0.05	2.67±0.58	0.33±0.58	0±0	0.33±0.58	0±0	46.086	0.442	6	0.59±0.10	2.33±0.82	0±0	0±0	0.17±0.41	0±0
	4	2.372	0.359	4	0.67±0.17	3.25±1.89	0±0	0±0	0.50±0.58	0±0	35.629	0.345	4	0.53±0.03	3.00±0.82	0.75±0.50	0±0	0±0	0±0
	5	2.933	0.480	4	0.62±0.11	3.50±1.00	0.50±0.58	0±0	0.50±0.58	0±0	50.724	0.518	5	0.53±0.03	2.80±1.10	0.40±0.55	0±0	0±0	0±0

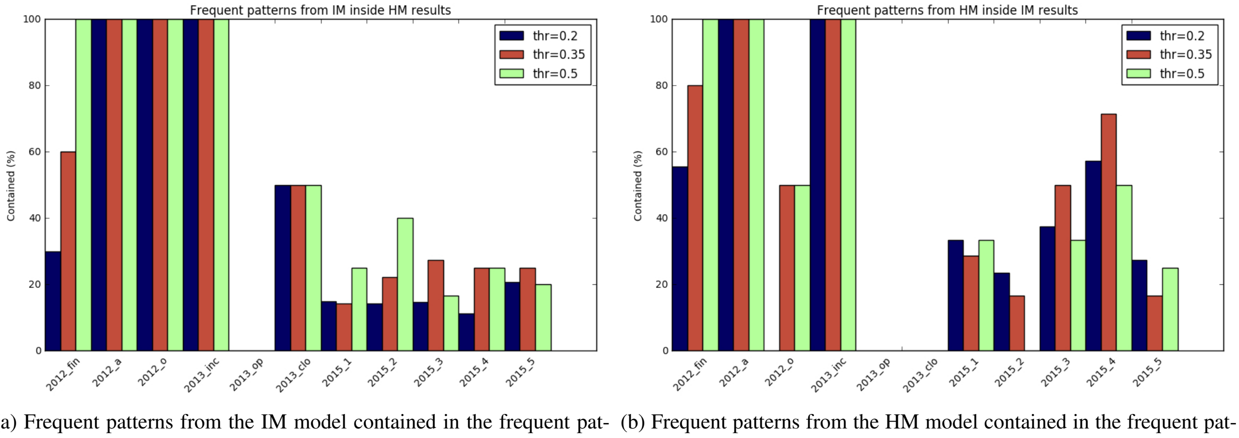
Table 10: Behavioral structure of the frequent patterns extracted with a threshold of 50% from the process models of the BPICs. It shows the information for the results with two process models of each log (Heuristics and Inductive). The information contains the runtime, the number of patterns and the distribution (average and standard deviation) of the frequency, the number of activities, sequences, choices, parallels and loops of each pattern. The missing results in the 2011 log with the IM's model are due to a non convergence of the algorithm, taking more than 5 hours to execute a few iterations.

Regarding the runtime, we can observe that most of the values are close to 5 seconds, with the exception of the more complex models (*BPIC 2011*, *BPIC 2012-fin*, *BPIC 2013-inc*, IM of all *BPIC 2015*) ranging from 2 to 6 minutes. Most of this time is spent on the preprocessing, which can be shared between executions with different thresholds, reducing the total runtime in real applications. Also, there is a significant difference in the algorithm runtime between models, specially for a threshold of 20%. These differences are due to the very different grades of complexity of the mined models. For higher thresholds, the differences weaken, as most of the structures are pruned in the first analysis, and the expansion step of WoMine does not consider them.

Besides, we have compared the number of patterns discovered for the same threshold for the HM and the IM models. With logs where the difference between the mined models is higher —2011 and 2015—, the number of retrieved patterns with more complex models (IM) is significantly higher. The algorithm builds more structures with these models and, consequently, extracts more patterns. This difference is attenuated when the threshold increases, because the patterns not represented in the HM models are patterns with low frequency. On the other hand, with simpler models —2012 and 2013—, the differences are less notable and the number of patterns extracted are almost the same.

In a more exhaustive comparison, we have analysed how many patterns extracted from one model are contained in the results of the other one (Fig. 16). With less complex logs —2012 and 2013— almost all of the patterns from the IM model are retrieved using the HM model. In contrast, with complex models, the set of patterns from the IM model is more complete than the set of patterns from the HM model. Usually, with a more complex model, more behaviour is examined and more patterns can be retrieved, with the penalty of a higher runtime. But increasing the complexity of a model might hide structures in the search of WoMine, causing the lost of frequent behaviour.

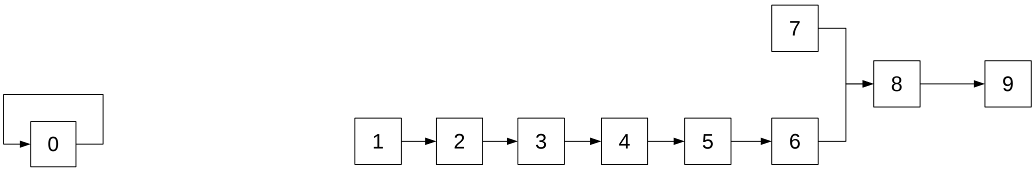
Fig. 17 shows two examples of patterns extracted by WoMine from the HM model of the BPIC 2011 log, which corresponds to a Dutch Academic Hospital. This model contains more than 623 activities and almost 1,500 arcs. Fig. 17a presents a pattern extracted from the model which appears in the 65% of the traces. This pattern is formed by a single activity, with a loop to itself. A frequent structure like this may warn the staff of the hospital about a possible error that is occurring in the process. It might also be a correct behaviour, and its detection may help the process manager not to free the resources used after this activity, because is very common to be executed more times. Fig. 17b shows another pattern formed by two sequences, joined by a choice. WoMine detects this pattern in the 22% of the traces. With this information, the process manager may try to optimize the subprocess, or schedule the resources to improve the execution of the process.



(a) Frequent patterns from the IM model contained in the frequent patterns set from the HM model.

(b) Frequent patterns from the HM model contained in the frequent patterns set from the IM model.

Figure 16: Percentage of patterns obtained from the model mined by a discovery algorithm that are contained into patterns from the mined model of the other algorithm.



(a) Frequent pattern (65.44%) extracted from the 2011 log with the Heuristics model. The real name of the activity is *'aanname laboratoriumonderzoek'* (assumption laboratory).

(b) Frequent pattern (22.65%) extracted from the 2011 log with the Heuristics model. The pattern is formed by two sequences joined by a choice (XOR-join). The real name of the activities are: 1: *'kalium potentiometrisch'* (potassium potentiometric); 2: *'sgot - asat kinetisch'* (Glutamic-oxalacetic transaminase); 3: *'sgpt - alat kinetisch'* (Glutamic-pyruvic transaminase); 4: *'melkzuurdehydrogenase -ldh-kinetisch'* (Lactic acid dehydrogenase); 5: *'bloedgroep abo en rhesusfactor'* (abo blood group and rhesus factor); 6: *'rhesusfactor d - centrifugeermethode – e'* (Rhesus factor d - Centrifuge method); 7: *'differentiele telling automatisch'* (differential count automatically); 8: *'leukocyten tellen elektronisch'* (leukocyte count electronic); 9: *'trombocyten tellen – elektronisch'* (platelet count electronic)

Figure 17: Two frequent patterns retrieved from the BPIC tests.

8. Conclusion and Future Work

We have presented WoMine, an algorithm designed to search frequent patterns in an already discovered process model. The proposal, based on a novel a priori algorithm, is able to find patterns with the most common control structures, including loops. We have compared WoMine with the state of the art approaches, showing that, although the other proposals fail for some of the models, WoMine always retrieves the correct frequent patterns. Moreover, we have also tested WoMine with complex real logs from the BPICs. Results show the importance of the frequent patterns to analyze and optimize the process model.

Regarding possible future work, the frequent behaviour extracted by WoMine might be used for many tasks. Particularly, the detection of outliers or anomalies might be done by giving a score to the traces depending on how much frequent behavior is contained in them, as be in [5] and [10]. Rating the traces of the log depending on the frequent behavior they contain can also be useful for simplification. A more simpler model can be mined from the log if the traces with less frequent behavior are removed. There are also possible applications in the field of decomposition using the frequent structures to decompose the process as is done in [20] or in [26].

Acknowledgments.

This research was funded by the Spanish Ministry of Economy and Competitiveness under grant TIN2017-84796-C2-1-R, and the Galician Ministry of Education, Culture and Universities under grant ED431G/08. These grants are co-funded by the European Regional Development Fund (ERDF/FEDER program). D. Chapela-Campa is supported by the Spanish Ministry of Education, under the FPU national plan (FPU16/04428).

References

References

[1] Agrawal, R., Imielinski, T., Swami, A. N., 1993. Mining association rules between sets of items in large databases. In: Buneman, P., Jajodia, S. (Eds.), Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD 1993), Washington, D.C. ACM Press, pp. 207–216.

[2] Agrawal, R., Srikant, R., 1995. Mining sequential patterns. In: Yu, P. S., Chen, A. L. P. (Eds.), Proceedings of the 11th International Conference on Data Engineering (ICDE 1995), Taipei, Taiwan. IEEE, pp. 3–14.

[3] Bui, D. B., Hadzic, F., Potdar, V., 2012. A framework for application of tree-structured data mining to process log analysis. In: Yin, H., Costa, J. A. F., Barreto, G. D. A. (Eds.), Proceedings of the 13th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2012), Natal, Brazil. Vol. 7435 of Lecture Notes in Computer Science. Springer, pp. 423–434.

[4] Buijs, J. C. A. M., van Dongen, B. F., van der Aalst, W. M. P., 2012. On the role of fitness, precision, generalization and simplicity in process discovery. In: Meersman, R., Panetto, H., Dillon, T. S., Rinderle-Ma, S., et al. (Eds.), Proceedings of the On the Move to Meaningful Internet Systems: Part I (OTM 2012), Rome, Italy. Vol. 7565 of Lecture Notes in Computer Science. Springer, pp. 305–322.

[5] Chiu, C., Yeh, C., Lee, Y., 2013. Frequent pattern based user behavior anomaly detection for cloud system. In: Proceedings of the 2013 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2013), Taipei, Taiwan. IEEE, pp. 61–66.

[6] de Medeiros, A. K. A., 2006. Genetic process mining. Ph.D. thesis, TUE : Department of Industrial Engineering and Innovation Sciences.

[7] de San Pedro, J., Carmona, J., Cortadella, J., 2015. Log-based simplification of process models. In: Motahari-Nezhad, H. R., Recker, J., Weidlich, M. (Eds.), Proceedings of the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria. Vol. 9253 of Lecture Notes in Computer Science. Springer, pp. 457–474.

[8] Desel, J., Reisig, W., 1996. Place or transition petri nets. In: Reisig, W., Rozenberg, G. (Eds.), Lectures on Petri Nets I: Basic Models, Advances in Petri Nets. Vol. 1491 of Lecture Notes in Computer Science. Springer, pp. 122–173.

[9] Fahland, D., van der Aalst, W. M. P., 2011. Simplifying mined process models: An approach based on unfoldings. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (Eds.), Proceedings of the 9th International Conference on Business Process Management (BPM 2011), Clermont-Ferrand, France. Vol. 6896 of Lecture Notes in Computer Science. Springer, pp. 362–378.

[10] Ghionna, L., Greco, G., Guzzo, A., Pontieri, L., 2008. Outlier detection techniques for process mining applications. In: An, A., Matwin, S., Ras, Z. W., Slezak, D. (Eds.), Proceedings of the 17th International Symposium on Foundations of Intelligent Systems (ISMIS 2008), Toronto, Canada. Vol. 4994 of Lecture Notes in Computer Science. Springer, pp. 150–159.

[11] Greco, G., Guzzo, A., Manco, G., Pontieri, L., Saccà, D., 2004. Mining constrained graphs: The case of workflow systems. In: Boulicaut, J., Raedt, L. D., Mannila, H. (Eds.), Proceedings of the 2004 European Workshop on Inductive Databases and Constraint Based Mining, Hinterzarten, Germany. Vol. 3848 of Lecture Notes in Computer Science. Springer, pp. 155–171.

[12] Greco, G., Guzzo, A., Pontieri, L., Saccà, D., 2004. Mining expressive process models by clustering workflow traces. In: Dai, H., Srikant, R., Zhang, C. (Eds.), Proceedings of the 8th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2004), Sydney, Australia. Vol. 3056 of Lecture Notes in Computer Science. Springer, pp. 52–62.

[13] Greco, G., Guzzo, A., Pontieri, L., Saccà, D., 2006. Discovering expressive process models by clustering log traces. IEEE Trans. Knowl. Data Eng. 18, 1010–1027.

[14] Günther, C. W., Rozinat, A., 2012. Disco: Discover your processes. In: Lohmann, N., Moser, S. (Eds.), Proceedings of the Demonstration Track of the 10th International Conference on Business Process Management (BPM 2012), Tallinn, Estonia. Vol. 940 of CEUR Workshop Proceedings. CEUR-WS, pp. 40–44.

[15] Han, J., Cheng, H., Xin, D., Yan, X., 2007. Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov. 15, 55–86.

[16] Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M., 2000. Freespan: frequent pattern-projected sequential pattern mining. In: Ramakrishnan, R., Stolfo, S. J., Bayardo, R. J., Parsa, I. (Eds.), Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2000), Boston, MA, USA. ACM, pp. 355–359.

[17] Leemans, M., van der Aalst, W. M. P., 2014. Discovery of frequent episodes in event logs. In: Ceravolo, P., Russo, B., Accorsi, R. (Eds.), Proceedings of the 4th International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA 2014), Milan, Italy. Vol. 237 of Lecture Notes in Business Information Processing. Springer, pp. 1–31.

[18] Leemans, S. J. J., Fahland, D., van der Aalst, W. M. P., 2013. Discovering block-structured process models from event logs - A constructive approach. In: Colom, J. M., Desel, J. (Eds.), Proceedings of the 34th International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2013), Milan, Italy. Vol. 7927 of Lecture Notes in Computer Science. Springer, pp. 311–329.

[19] Mannila, H., Toivonen, H., Verkamo, A. I., 1997. Discovery of frequent episodes in event sequences. Data Min. Knowl. Discov. 1, 259–289.

[20] Munoz-Gama, J., Carmona, J., van der Aalst, W. M. P., 2014. Single-entry single-exit decomposed conformance checking. Inf. Syst. 46, 102–122.

[21] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M., 2001. Prefixspan: Mining sequential patterns by prefix-projected growth. In: Georgakopoulos, D., Buchmann, A. (Eds.), Proceedings of the 17th International Conference on Data Engineering (ICDE 2001), Heidelberg, Germany. IEEE, pp. 215–224.

[22] Song, M., Günther, C. W., van der Aalst, W. M. P., 2008. Trace clustering in process mining. In: Ardagna, D., Mecella, M., Yang, J. (Eds.), Proceedings of the 2008 International Workshops on Business Process Management (BPM 2008), Milano, Italy. Vol. 17 of Lecture Notes in Business Information Processing. Springer, pp. 109–120.

[23] Steeman, W., 2013. BPI Challenge 2013.
URL https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07

[24] Tax, N., Sidorova, N., Haakma, R., van der Aalst, W. M. P., 2016. Mining local process models. J. Innov. Digit. Ecosyst. 3, 183–196.

[25] van der Aalst, W. M. P., 2011. Process Mining - Discovery, Conformance and Enhancement of Business Processes, 1st Edition. Springer.

[26] van der Aalst, W. M. P., 2012. Decomposing process mining problems using passages. In: Haddad, S., Pomello, L. (Eds.), Proceedings of the 33rd International Conference on Application and Theory of Petri Nets (PETRI NETS 2012), Hamburg, Germany. Vol. 7347 of Lecture Notes in Computer Science. Springer, pp. 72–91.

[27] van der Aalst, W. M. P., Adriansyah, A., van Dongen, B. F., 2011. Causal nets: A modeling language tailored towards process discovery. In: Katoen, J., König, B. (Eds.), Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR 2011), Aachen, Germany. Vol. 6901 of Lecture Notes in Computer Science. Springer, pp. 28–42.

[28] van der Aalst, W. M. P., de Medeiros, A. K. A., Weijters, A. J. M. M., 2005. Genetic process mining. In: Ciardo, G., Darondeau, P. (Eds.), Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005), Miami, USA. Vol. 3536 of Lecture Notes in Computer Science. Springer, pp. 48–69.

[29] Van Dongen, B., 2011. Real-life event logs - hospital log.
URL https://doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54

[30] Van Dongen, B., 2012. BPI Challenge 2012.
URL https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

[31] Van Dongen, B., 2015. BPI Challenge 2015.
URL https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1

[32] van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., van der Aalst, W. M. P., 2005. The prom framework: A

new era in process mining tool support. In: Ciardo, G., Darondeau, P. (Eds.), Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005), Miami, USA. Vol. 3536 of Lecture Notes in Computer Science. Springer, pp. 444–454.

[33] Vázquez-Barreiros, B., Lama, M., Mucientes, M., Vidal, J. C., 2014. Softlearn: A process mining platform for the discovery of learning paths. In: Chen, N.-S., Huang, R., Kinshuk, Li, Y., Sampson, D. G. (Eds.), Proceedings of the IEEE 14th International Conference on Advanced Learning Technologies (ICALT 2014), Athens, Greece. IEEE, pp. 373–375.

[34] Vázquez-Barreiros, B., Mucientes, M., Lama, M., 2015. Prodigen: Mining complete, precise and minimal structure process models with a genetic algorithm. Inf. Sci. 294, 315–333.

[35] Weijters, A. J. M. M., van der Aalst, W. M. P., de Medeiros, A. K. A., 2006. Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Technical Report WP 166, 1–34.

[36] Zaki, M. J., 2001. SPADE: an efficient algorithm for mining frequent sequences. Mach. Learn. 42, 31–60.