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* [23]. 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.

There are four quality dimensions to measure *how good* a process model is: *fitness replay*, *precision*, *generalization* and *simplicity*. Regarding the latter, discovering a complex process model, i.e., a hardly readable process model, can totally hinder its quality [5] making difficult the retrieval of behavioural information. Different techniques have been proposed to tackle this problem: the simplification of already mined models [5, 7], the search of simpler structures in the logs [15, 17, 22], or the clusterization of the log into smaller and more homogeneous subsets of traces

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

to discover different models within the same process [9, 10, 18]. Although these techniques improve the understandability of the process models, for real processes the model structure remains complex, being difficult to understand by users.

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 [15, 22], e.g., a structure executed 1,000 times; or *ii)* its high repetition in the traces of the log [3, 8], 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 [28], 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 [8], 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 tasks, 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 [11]. 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 [12], can build frequent patterns based just on the logs, searching in them for frequent sequences of tasks [13, 14, 31]. Improving this search, episode mining techniques focus their search in frequent, and more complex structures such as parallels [15, 17]. With a different approach, the *w*-find algorithm [8] 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. [22] 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 tasks[2].

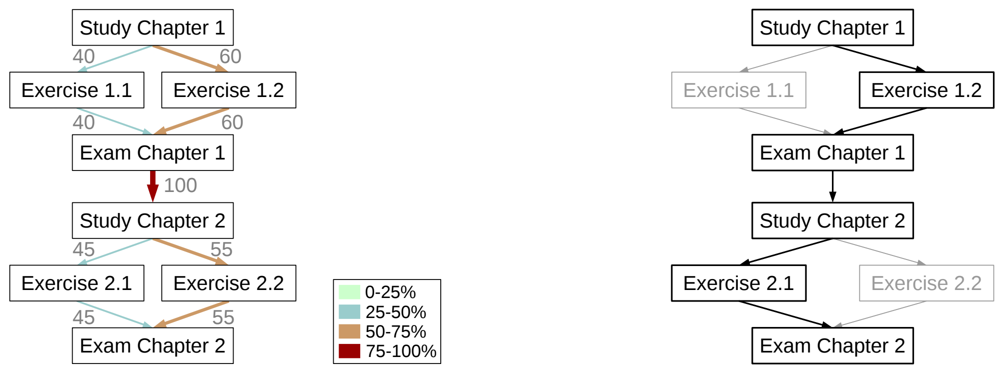
The heat maps approach performs a highlight of the arcs and tasks of a process model depending on their individual frequency, i.e., the number of executions. To obtain the frequent patterns of a model using heat maps, the arcs which frequency exceeds a defined threshold are retrieved. The problem is that the individual frequency does not consider the causality between tasks. 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. 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, 14, 15, 17]. Some of these techniques are based on sequential pattern mining (SPM), and search subsequences of tasks with a high

[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 tasks to the tasks of a selection (choice) where one of the branches has no tasks, leaving the other as optional.



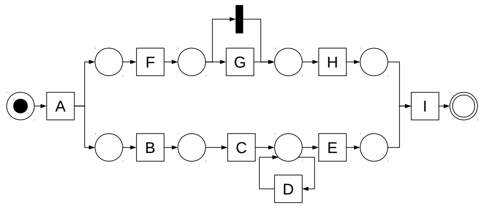
(a) Heat Maps example: C-net with the arcs highlighted depending on their absolute frequency.

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

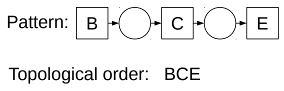
Figure 1: Process model of a simple process in the education domain.

frequency in large sequences [2, 14]. 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 [14], 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 tasks. Structures like concurrences or selections are treated as different sequences depending on the order of the tasks. Also, in a retrieved frequent sequence, the execution of the i-th task is not ensured to be caused by the $i-1$-th task in all the occurrences in the log —SPM only checks if the tasks 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. [17]. An episode is a collection of tasks 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, Leemans et al. have designed an algorithm [15] to extract association rules from frequent episodes measuring their frequency with the instances of the process, instead of using windows in the task 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 tasks presented in it. For the same frequent behaviour, the algorithm extracts various patterns with the same tasks, but with different relations among them, making difficult the extraction of information.



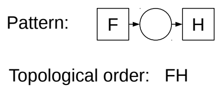
(a) Petri net with two parallel branches, one with a loop and the other one with an optional task.



(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.

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 [8] —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 tasks 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 task 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 [22]. 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 tasks, and adding different relations with other tasks. 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 tasks 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 tasks, 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 tasks. The approaches based on episode mining [15, 17] and frequent pattern mining [8] 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 tasks. 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 tasks 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 tasks, ensuring

		Mine from model	Expl. proc. instances	Mine sequences	Mine parallels	Mine choices	Mine loops
Sequential Pattern Mining - Based Algorithms [14]		-	-	+	-	-	-
Mannila's Episode mining [17]		-	-	+	+	-	-
Bui's Tree Mining [3]		-	+	+	-	-	-
Leemans' Episode discovery [15]		-	+	+	+	-	-
w-find [8]		+	+	+	+	-	-
Tax's Local Process Model [22]		-	+	+	+	±	±
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.

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 [6] due to its higher comprehensibility, and the easiness to explain the behaviour of the execution. Nevertheless, our algorithm represents the process with a Causal Matrix (Def. 1).

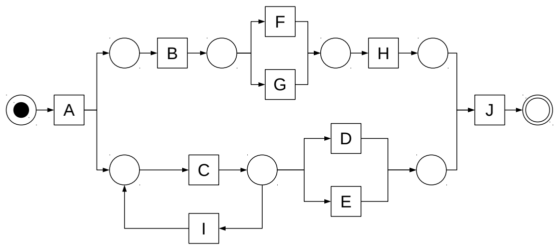
Definition 1 (Causal matrix). A Causal matrix is a tuple (T, I, O) where:

T is a finite set of tasks.

I : $T \to \mathbb{P}(\mathbb{P}(T))$ is the input condition function, where $\mathbb{P}(X)$ denotes the powerset of some set X.

O : $T \to \mathbb{P}(\mathbb{P}(T))$ is the output condition function.

For a task or activity $\alpha \in T$, $I(\alpha)$ denotes a set of sets of tasks representing the inputs of α. Each set $\Phi \in I(\alpha)$ corresponds to a choice in the inputs of α —$|I(\alpha)| > 1$ represents a selection, and $|\Phi| > 1$ denotes a parallel input path. In the same way, $O(\alpha)$ denotes the set of sets of tasks representing the outputs of α —$|\Theta| > 1 : \Theta \in O(\alpha)$ denotes a parallel output path. Fig. 3 shows an example.



(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.

Task	I(Task)	O(Task)
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 matrix of the process model in Fig. 3a.

Figure 3: Example to show the internal representation of the process models in WoMine. The inputs of task 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.

Definition 2 (Trace). Let T be the set of tasks of a process model, and ε an event —the execution of a task $\alpha \in T$. A trace is a list (sequence) $\langle \tau = \varepsilon_1, ..., \varepsilon_n \rangle$ 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, the events only specify the name of the task, but usually, the event logs store more information as timestamps, resources, etc.

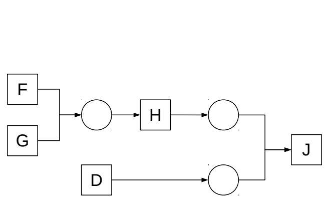
Definition 4 (Pattern). Let $C = (T, I, O)$ be a causal matrix representing a process model W. A connected subgraph $P = (T', I', O')$ is a pattern of W if and only if:

$T' \subseteq T$

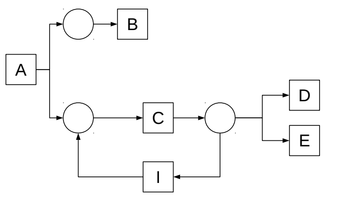
$|I'| = |O'| = |T'|$

$\forall \alpha \in T' \rightarrow I'(\alpha) \subseteq I(\alpha), O'(\alpha) \subseteq O(\alpha)$

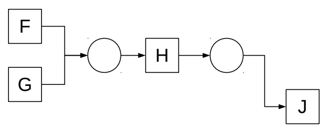
A *pattern* (Def. 4) is a subgraph of the process model that represents the behaviour of a part of the process. For each task α in the pattern, its inputs, $I'(\alpha)$, must be a subset of $I(\alpha)$ in the model it belongs to; and the outputs, $O'(\alpha)$, must be also a subset of $O(\alpha)$ in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



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

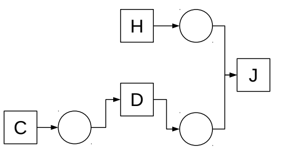


(b) Valid pattern with a parallel, a selection and a loop.

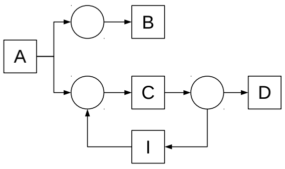


(c) Invalid pattern. Task `J` has some incomplete input combinations.

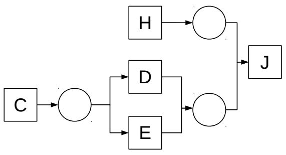
Figure 4: Examples of valid and invalid patterns of the process model shown in Fig. 3. All tasks have as connections a subset of their connections in the causal matrix of Table 3c. For instance, $I'(J)$ is equal to {{`H,D`}}, which is a subset of $I(J)$, {{`H,D`}, {`H,E`}}. This makes the structure a valid pattern. Meanwhile, the structure shown in Fig. 4c is a wrong pattern, because task `J` has some incomplete input combinations —{{`H`}} $\not\subseteq$ {{`H,D`}, {`H,E`}}.



(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. Tasks `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.

Definition 5 (Simple Pattern). A pattern $P = (T', I', O')$ is a Simple Pattern when:

$\forall \alpha \in T' \rightarrow [\exists! \Phi \in I'(\alpha) \colon \Phi \nsubseteq R^{+}(\alpha)] \vee [\forall \Phi \in I'(\alpha) \colon \Phi \subseteq R^{+}(\alpha)]$

$\forall \alpha \in T' \rightarrow [\exists! \Theta \in O'(\alpha) \colon \Theta \nsubseteq R^{-}(\alpha)] \vee [\forall \Theta \in O'(\alpha) \colon \Theta \subseteq R^{-}(\alpha)]$

Being $R^{+}(\alpha)$ the set of successors of a task α, i.e., the set of tasks reachable from α; and $R^{-}(\alpha)$ the set of predecessors of a task α, i.e., the set of tasks that can reach α.

The Simple Patterns (Def. 5) are those patterns which behaviour can be executed, entirely, in one instance. If a task has a selection, it must be able to execute each path in the same instance. For this, the inputs of each task α must have all tasks reachable from α except, at most, the tasks 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 task of the process model belongs to, at least, one Minimal Pattern. The M-pattern of a task 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 the inputs or outputs, there must be an M-pattern with each parallel path. Given a causal matrix $C = (T, I, O)$ representing a process model W and a task $\alpha_1 \in T$, a simple pattern $P = (T', I', O')$ is a Minimal Pattern of α_1 if all the following rules are satisfied:

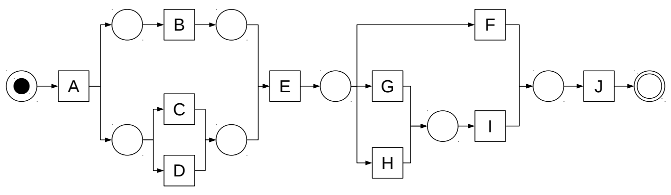
$|I(\alpha_1)| > 1, \forall \Phi \in I(\alpha_1) \colon |\Phi| = 1 \rightarrow I'(\alpha_1) = \emptyset$ $\quad$ $|O(\alpha_1)| > 1, \forall \Theta \in O(\alpha_1) \colon |\Theta| = 1 \rightarrow O'(\alpha_1) = \emptyset$

$\exists \Phi \in I(\alpha_1) \colon |\Phi| > 1 \rightarrow |I'(\alpha_1)| = \{\Phi\}$ $\quad$ $\exists \Theta \in O(\alpha_1) \colon |\Theta| > 1 \rightarrow |O'(\alpha_1)| = \{\Theta\}$

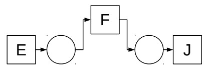
$\forall \alpha \in T' \colon |I(\alpha)| = 1 \rightarrow I'(\alpha) = I(\alpha)$ $\quad$ $\forall \alpha \in T' \colon |O(\alpha)| = 1 \rightarrow O'(\alpha) = O(\alpha)$

$\forall \alpha \in T', \alpha \neq \alpha_1, \alpha \notin R^{+}(\alpha_1), |I(\alpha)| > 1 \rightarrow I'(\alpha) = \emptyset$ $\quad$ $\forall \alpha \in T', \alpha \neq \alpha_1, \alpha \notin R^{-}(\alpha_1), |O(\alpha)| > 1 \rightarrow O'(\alpha) = \emptyset$

In WoMine each task has associated, at least, one M-pattern. For a task α_1, its M-patterns include: *i)* the structure executed always when α_1 is executed —1-path inputs and 1-path outputs—, and *ii)* the parallel connections that α_1

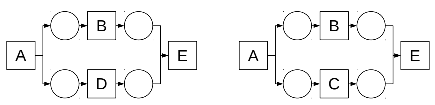


(a) Petri net of the process model.



(b) M-pattern of F.

(c) M-pattern of J.



(d) M-patterns of A.

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

has. If the inputs of α_1 have a selection and no parallel choices, the inputs of α_1 in the M-pattern must be empty. If α_1 has a parallel connection in the inputs, there must be an M-pattern with the parallel connection. For each task α in the M-pattern having only one possible input path, α must have the same path in the M-pattern —because this path will be always executed when α is executed. Finally, for all the tasks not being reached from α_1 —not being in the successors $R^+(\alpha_1)$—, except α_1, if they have a selection in the inputs, they must have its inputs empty in the M-pattern. All these rules are similar for the outputs, but with the predecessors instead of the successors.

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.

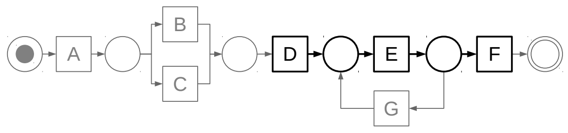
Definition 7 (Candidate Arcs). Let $C = (T, I, O)$ be a causal matrix representing a process model W. An arc $\langle \alpha_i \rightarrow \alpha_j \rangle \colon \alpha_i, \alpha_j \in T$ is part of the $A^<$ set, i.e., a candidate arc, if and only if:

$\forall \Theta \in O(\alpha_i)\colon \Theta = \{\alpha_j\} \vee \alpha_j \notin \Theta$

$\forall \Phi \in I(\alpha_j)\colon \Phi = \{\alpha_i\} \vee \alpha_i \notin \Phi$

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 $\langle A \rightarrow B \rangle$, $\langle A \rightarrow C \rangle$, $\langle A \rightarrow D \rangle$, $\langle B \rightarrow E \rangle$, $\langle C \rightarrow E \rangle$ and $\langle D \rightarrow E \rangle$ are included in the $A^<$ set.

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 execution of the trace in the process model contains the execution of the pattern, i.e., all arcs and tasks of SP are executed in a correct order, and each task fires the execution of its outputs in the pattern.



Trace:

`A B D E G E F`

Executed arcs:

$\langle A \to B\rangle, \langle B \to D\rangle,$
$\langle D \to E\rangle, \langle E \to G\rangle,$
$\langle G \to E\rangle, \langle E \to F\rangle$

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.

To check the compliance it is important that, while the pattern is being executed, each of its tasks triggers only the execution of its outputs. There are cases where all the arcs and tasks 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 tasks 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 (Frequecy of a 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, given by:

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

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[3] [8] starting with the frequent minimal patterns (Def. 6) of the model. In

[3]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.

this search, there is a 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 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

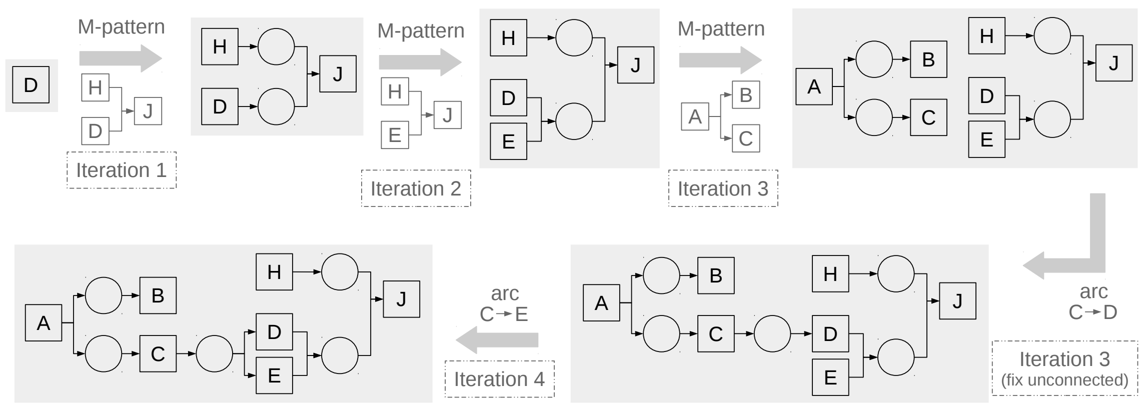


Figure 8: Example of the expansion of a pattern in 4 iterations. The process starts with the *M*-pattern of `D`, being expanded with an *M*-pattern of `J` in the first iteration. In the second iteration the new pattern is extended with another *M*-pattern of `J`. The third iteration shows an addition 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.

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 the $p \in currentPatterns$ do
            $candidatePatterns \leftarrow candidatePatterns \cup$ addFrequentArcs ($p$)
            $complementaryM \leftarrow \{m \mid m \in M, m \notin p\}$
            forall the $m \in complementaryM$ 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 the $arc \in frequentArcs$ 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 the $arc \in freqArcs$ 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 $patterns$
```

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.

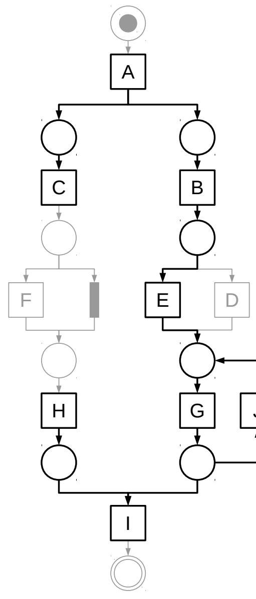
Algorithm 2. Check if a given pattern is executed more times than a threshold.

```
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.
Algorithm isFrequentPattern (pattern, T, threshold)
    simplePatterns ← generate the simple patterns of pattern
    frequencies ← $\emptyset$
    forall the simplePattern $\in$ simplePatterns do
        frequencies ← frequencies $\cup$ getPatternFrequency (simplePattern, T)
    end
    $minFreq \leftarrow 0$
    if frequencies.length $> 0$ then
        $minFreq$ ← minimum of frequencies
    end
    $realFreq \leftarrow minFreq/T.length$
    return $realFreq \geq threshold$
Function getPatternFrequency (pattern, T)
    executed ← 0
    forall the trace $\in T$ do
        if isTraceCompliant (pattern, trace) then
            executed ← executed + 1
        end
    end
    return executed
Function isTraceCompliant (pattern, trace)
    forall the task $\in$ trace do
        Execute task in the process model
        sources ← get the tasks that fired the execution of task
        simulateExecutionInPattern (sources, task, pattern)
        if pattern has been successfully executed then
            return true
        end
    end
    return false
```

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 threshold 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 tasks in the trace (Alg. 2:22), simulating its execution in the model, and retrieving the tasks 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 task —the fired one— and the tasks that have fired it —the firing tasks, retrieved by the simulation—, the executed tasks 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 tasks* empty. The first step (#1) executes `A`. There are



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

Trace: `A B E G J G C F H I`			
Initial tasks: `[A, H]` **End tasks**: `[C, I]`			
#	**executed task**	**executed arcs**	**last executed tasks**
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 task'* is the task currently executed; *'executed arcs'* is the set with the arcs belonging to the pattern which execution was correctly saved; *'Last executed tasks'* is the set of tasks 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.

no firing tasks because `A` is the initial task of the process model. As `A` is also one of the initial tasks of the pattern, it is saved correctly in the *last executed tasks* set.

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

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

At step #8, only the source task 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 tasks 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 tasks*.

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

Finally (#10), I has two firing tasks and, thus, two arcs are executed. In both cases, the source task of the arcs —G and H— is in the *last executed tasks* set, and the arc is in the pattern. Thus, a simple addition of I to the *last executed tasks* 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 tasks* set corresponds with the end tasks 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.

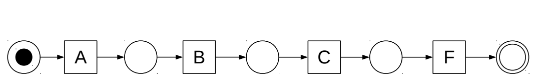
The process of saving the executed arcs and tasks has to be restarted when the executed arc is disrupting the execution of the pattern. For instance, in step #3, if the arc $\langle B \rightarrow D\rangle$ was executed instead of $\langle B \rightarrow E\rangle$, this would cause this saving process to go back by removing the arcs and tasks 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[4].

6. Simplifying the Result Set of Patterns

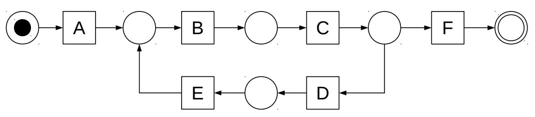
The result set of the a priori search has a high redundancy. This is because there are patterns in the k-th iteration which are expanded and thus are subpatterns of those in the $k + 1$-th iteration. A naive approach to reduce the redundancy generated by the expansion might be to remove the patterns 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 tasks 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.

[4]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.

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

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

Definition 12 (Endloop arc). Given a process model composed by a set of tasks T, and a set of arcs E, an *Endloop* arc $e \in E$ 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 task 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 tasks of the pattern and goes forward until it reaches a task with more than one output. When this happens, an analysis trying to close a loop —reach the current task again— is thrown for each output arc. The analysis goes forward through non-Startloop arcs and finishes when it reaches the current task or the end of the pattern. If the analysis reaches the starting task, 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).

#	task under analysis	outputs	subanalysis			action
			possible Startloop arc	current task	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 tasks, end of first phase

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

Endloop search

This search starts with the target task of each Startloop detected in the same iteration. For each task, it goes forward through its output arcs by analysing their target tasks. The analysis continues until a task 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 task going backwards through the $\langle A \rightarrow B\rangle$ arc (#7). Therefore, the current arc, $\langle E \rightarrow B\rangle$, is marked as Endloop.

#	Startloop	possible Endloop	task 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 task; *'task to reach the start from'* is the task from which the search is trying to reach the start in that step; *'pending to analyse'* is the set of input tasks that have not been analysed in the search for the start of the pattern; *'input under analysis'* is the task 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.

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 the arc ∈ startloopArcs do
            startTask ← arc.target
            if arriveStartWithoutLoop (startTask, startloopArcs) then
                set arc as Endloop
            else
                continueWithOutputs (startTask, startloopArcs)
            end
        end
    while startloopArcs ≠ ∅
    return p
Function searchStartloopArcs (p, previousStartloops)
    initialTasks ← ∅
    if previousStartloops = ∅ then
        initialTasks ← start tasks of p
    else
        initialTasks ← targets of arcs in previousStartloops
    end
    forall the task ∈ initialTasks do
        Go forward through the outputs while there is only one
        forall the output ∈ sorted task.outputs do
            analizeSplit (task, output)
        end
    end
    return the set of new Startloops
Function continueWithOutputs (previousTask, startloopArcs)
    forall the output ∈ previousTask.outputs do
        if arriveStartWithoutLoop (output, startloopArcs) then
            set ⟨previousTask → output⟩ as Endloop
        else
            if output has not been explored then
                continueWithOutputs (output, startloopArcs)
            end
        end
    end
```

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.

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 [4], 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 and 8GB of RAM (1600 MHz)[5].

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

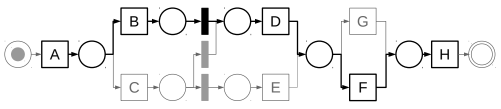
In this comparison, 5 process models with the most common control structures have been used. These models present scenarios where WoMine is able to retrieve frequent patterns, while any of the other techniques fails. For each model, two Petri nets will be presented: one with a highlighted frequent pattern 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.

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

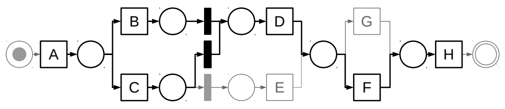
Table 4: Comparison between WoMine and other state of the art techniques for 5 process models: '+' stands for a correct frequent pattern extraction; '-' stands for a non extraction of the frequent pattern, and '±' stands for an incorrect extraction of the frequent pattern (similar but wrong structure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

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

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.



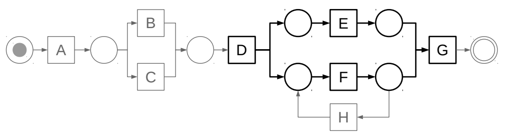
(a) WoMine pattern (highlighted) for a threshold of 40%.



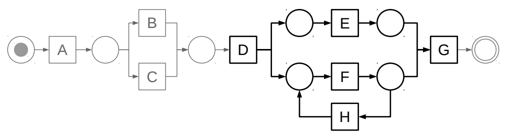
(b) Heat maps pattern (highlighted) for a threshold of 40. The pattern includes infrequent paths, for instance `A-C-D-H-F`. These arcs are frequent individually, but not the sequence itself. This happens because the executions of $\langle A \rightarrow C \rangle$ are distributed in traces through `D-G`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

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

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 a task 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 tasks but different relations, this pattern with a 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`.



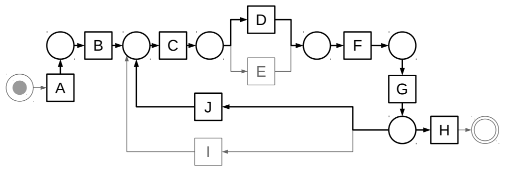
(a) WoMine pattern (highlighted) for a threshold of 70%. The traces compliant to this pattern are those where the loop is not executed.



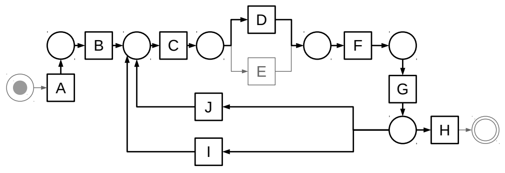
(b) Heat maps pattern (highlighted) for a threshold of 70. The parallel structure with the loop is not frequent. The loop is executed several times in the same trace, increasing its absolute frequency, but not the frequency of the whole pattern.

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

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 tasks that it checks. PrefixSpan gets the pattern retrieved by WoMine replacing the loop with a sequence with one repetition, i.e., duplicating the tasks 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.



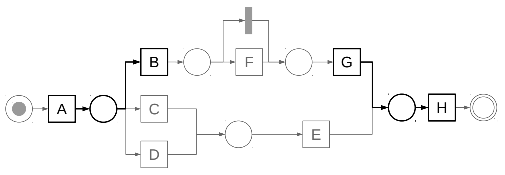
(a) WoMine pattern (highlighted) for a threshold of 55%. This structure corresponds the instances of the process model where `E` and `I` are not executed, and the loop of `J` is executed at least one time.



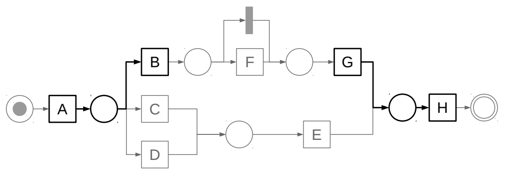
(b) Heat maps pattern (highlighted) for a threshold of 100. The arcs of the `I` loop are individually frequent, but the highlighted structure is not.

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 tasks. The loops give the model the ability to execute both branches of the selection in the same trace, and more than once.

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.



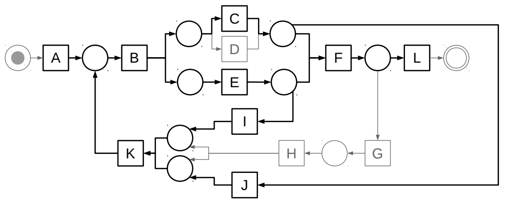
(a) WoMine pattern (highlighted) for a threshold of 65%.



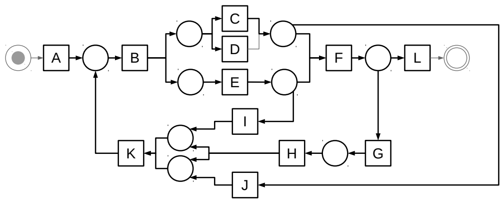
(b) Results of the heat maps approach applied with a threshold of 65. The result is correct and agrees with WoMine's.

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

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



(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 task, and one of them starting in a parallel structure.

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 tasks. 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 task causes the firing of another task. 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 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 tasks. 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 tasks 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 tasks, and containing loops, parallelisms, selections, etc. A more detailed description of the behavioral structures of these process models can be found in [4, 29]. Furthermore, as WoMine takes as starting point a process model and an event log, we used ProDiGen [29] 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 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 tasks for a threshold of 40%, and two patterns for 60% but with an average of 2.5 tasks per pattern.

	Threshold : 40%								
	runtime (secs)		#patt	frequency	#tasks	#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 [4]. The values show the runtimes for the preprocessing and for the algorithm; the number of patterns retrieved; the average and standard deviation of the frequency, number of tasks, number of sequences, number of choices, number of parallels and number of loops per pattern.

	Threshold : 60%								
	runtime (secs)		#patt	frequency	#tasks	#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	#tasks	#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%.

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) [24] and, on the other hand, to analyze the influence of the model in the retrieved patterns.

Table 7a shows some statistics of the BPIC logs [19, 20, 21, 24, 25, 26]. These logs have been mined with two of the most popular discovery algorithms, the Heuristics Miner (HM) [30] and the Inductive Miner (IM) [16]. Table 7b presents the characteristics of the mined models, which have been generated using ProM [27]. 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. Table 8 shows 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 logs *a* and *o* from BPIC 2012 have been generated after a filtering in the log, maintaining the traces with tasks of the categories A and O, respectively.

		Heuristics Miner		Inductive Miner	
		#tasks	#arcs	#tasks	#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 tasks 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	#tasks	#sequences	#choices	#parallels	#loops	runtime (secs)		#patt	frequency	#tasks	#sequences	#choices	#parallels	#loops
		pre	alg								pre	alg							
2011		5.847	289.127	20	0.25±0.08	5.65±4.30	0.70±0.66	0.80±1.15	0.25±0.44	0.40±0.50	-	-	-	-	-	-	-	-	-
2012	fin	214.118	6.748	7	0.29±0.05	3.14±2.91	0.29±0.49	0.14±0.38	0±0	0.29±0.49	216.847	52.493	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	0.014	0.029	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0	0.014	0.011	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0
	o	0.068	0.171	2	0.24±0.01	5.50±0.71	0±0	1.00±1.41	1.50±0.71	1.00±1.41	0.077	0.008	2	0.75±0.35	2.00±0.00	0±0	0±0	0±0	0±0
2013	inc	26.141	44.621	6	0.25±0.06	5.33±2.25	0±0	1.17±0.98	0±0	0.67±0.82	24.650	44.393	6	0.25±0.06	5.33±2.25	0±0	1.17±0.98	0±0	0.67±0.82
	clo	0.071	1.295	4	0.24±0.01	4.50±1.29	0±0	0.75±0.50	0±0	0.25±0.50	0.082	0.017	2	0.76±0.34	1.50±0.71	0±0	0±0	0±0	0.50±0.71
	op	0.021	0.077	2	0.21±0.00	4.00±0.00	0.50±0.71	1.00±1.41	0±0	0±0	0.023	0.017	1	0.30±0.00	1.00±0.00	0±0	0±0	0±0	1.00±0.00
2015	1	2.200	0.928	14	0.32±0.11	2.57±0.76	0.21±0.43	0±0	0.21±0.43	0±0	108.069	145.077	19	0.24±0.05	4.79±2.74	0.32±0.48	1.05±1.18	0.11±0.32	0±0
	2	0.961	1.298	15	0.28±0.14	3.27±1.58	0.27±0.46	0.07±0.26	0.47±0.64	0±0	90.468	80.182	26	0.24±0.04	3.27±2.07	0.31±0.47	0.42±0.76	0±0	0±0
	3	3.133	1.640	14	0.31±0.11	2.50±0.94	0.07±0.27	0.07±0.27	0.14±0.36	0±0	105.733	345.460	30	0.23±0.04	4.87±2.47	0.37±0.49	1.30±1.29	0.10±0.40	0±0
	4	1.532	1.976	12	0.35±0.20	4.00±2.37	0.17±0.39	0.08±0.29	0.50±0.67	0±0	77.010	9.934	20	0.24±0.03	4.15±2.91	0.55±0.60	0.45±0.83	0±0	0±0
	5	2.008	2.583	9	0.27±0.07	4.56±1.81	0.44±0.53	0.22±0.44	0.56±0.53	0±0	131.711	11.093	21	0.24±0.03	3.86±2.39	0.43±0.51	0.48±0.75	0±0	0±0

		Threshold : 35%																			
		Heuristics Miner									Inductive Miner									
		runtime (secs)		#patt	frequency	#tasks	#sequences	#choices	#parallels	#loops	runtime (secs)		#patt	frequency	#tasks	#sequences	#choices	#parallels	#loops
		pre	alg								pre	alg							
2011		5.847	4.602	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	214.118	0.994	4	0.38±0.01	4.00±2.45	0.50±0.58	0.25±0.50	0.25±0.50	0±0	216.847	2.215	4	0.38±0.01	4.25±2.87	0.50±1.00	0.25±0.50	1.25±2.50	0±0
	a	0.014	0.001	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0	0.014	0.001	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0
	o	0.068	0.012	2	0.75±0.35	3.50±2.12	0.50±0.71	0±0	0±0	0±0	0.077	0.005	2	0.75±0.35	2.00±0.00	0±0	0±0	0±0	0±0
2013	inc	26.141	1.384	2	0.43±0.09	4.00±1.41	0.50±0.71	0.50±0.71	0±0	1.00±1.41	24.650	1.495	2	0.43±0.09	4.00±1.41	0.50±0.71	0.50±0.71	0±0	1.00±1.41
	clo	0.071	0.023	2	0.87±0.10	2.50±0.71	0.50±0.71	0±0	0±0	0±0	0.082	0.005	2	0.76±0.34	1.50±0.71	0±0	0±0	0±0	0.50±0.71
	op	0.021	0.005	2	0.62±0.38	2.50±0.71	0.50±0.71	0±0	0±0	0±0	0.023	0.000	0	0±0	0±0	0±0	0±0	0±0	0±0
2015	1	2.200	0.452	7	0.48±0.10	2.43±0.79	0.14±0.38	0±0	0.14±0.38	0±0	108.069	1.649	7	0.49±0.10	2.57±1.13	0.14±0.38	0±0	0.14±0.38	0±0
	2	0.961	0.445	6	0.50±0.14	3.67±1.86	0.17±0.41	0±0	0.67±0.52	0±0	90.468	2.654	8	0.40±0.06	2.75±1.04	0.50±0.53	0.12±0.35	0±0	0±0
	3	3.133	0.519	6	0.43±0.11	2.67±0.82	0.33±0.52	0±0	0.17±0.41	0±0	105.733	2.079	11	0.47±0.11	2.27±0.65	0.09±0.30	0±0	0.09±0.30	0±0
	4	1.532	0.560	7	0.53±0.19	2.57±1.51	0±0	0±0	0.14±0.38	0±0	77.010	1.546	8	0.40±0.06	3.50±1.41	0.62±0.52	0±0	0±0	0±0
	5	2.008	0.921	6	0.49±0.17	4.00±1.67	0.33±0.52	0±0	0.67±0.52	0±0	131.711	2.588	8	0.38±0.03	3.00±1.41	0.38±0.52	0±0	0±0	0±0

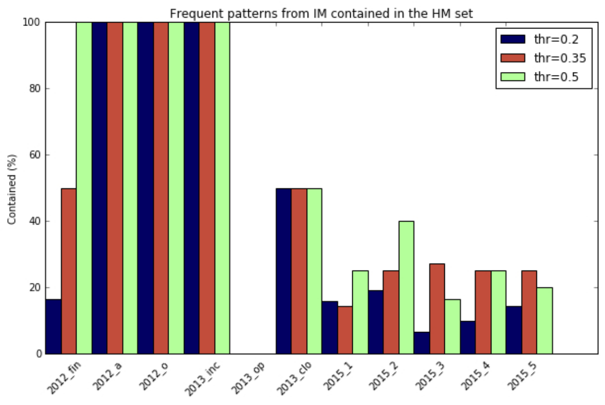
		Threshold : 50%																			
		Heuristics Miner									Inductive Miner									
		runtime (secs)		#patt	frequency	#tasks	#sequences	#choices	#parallels	#loops	runtime (secs)		#patt	frequency	#tasks	#sequences	#choices	#parallels	#loops
		pre	alg								pre	alg							
2011		5.847	0.914	9	0.53±0.02	2.44±1.01	0.33±0.50	0±0	0±0	0.22±0.44	-	-	-	-	-	-	-	-	-
2012	fin	214.118	0.209	2	0.78±0.31	2.50±0.71	0.50±0.71	0±0	0±0	0±0	216.847	0.170	2	0.78±0.31	2.50±0.71	0.50±0.71	0±0	0±0	0±0
	a	0.014	0.001	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0	0.014	0.001	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0
	o	0.068	0.012	2	0.75±0.35	3.50±2.12	0.50±0.71	0±0	0±0	0±0	0.077	0.002	2	0.75±0.35	2.00±0.00	0±0	0±0	0±0	0±0
2013	inc	26.141	0.780	4	0.67±0.15	2.75±0.50	0.25±0.50	0.50±0.58	0±0	0.25±0.50	24.650	0.755	4	0.67±0.15	2.75±0.50	0.25±0.50	0.50±0.58	0±0	0.25±0.50
	clo	0.071	0.021	2	0.87±0.10	2.50±0.71	0.50±0.71	0±0	0±0	0±0	0.082	0.005	2	0.76±0.34	1.50±0.71	0±0	0±0	0±0	0.50±0.71
	op	0.021	0.001	1	0.89±0.00	2.00±0.00	0±0	0±0	0±0	0±0	0.023	0.000	0	0±0	0±0	0±0	0±0	0±0	0±0
2015	1	2.200	0.241	3	0.63±0.03	2.67±0.58	0.33±0.58	0±0	0.33±0.58	0±0	108.069	1.970	4	0.61±0.06	2.75±1.50	0±0	0±0	0.25±0.50	0±0
	2	0.961	0.340	5	0.62±0.08	3.80±1.10	0±0	0±0	1.00±0.00	0±0	90.468	1.580	5	0.58±0.08	2.20±0.45	0.20±0.45	0±0	0±0	0±0
	3	3.133	0.255	3	0.68±0.05	2.67±0.58	0.33±0.58	0±0	0.33±0.58	0±0	105.733	1.353	6	0.59±0.10	2.33±0.82	0±0	0±0	0.17±0.41	0±0
	4	1.532	0.324	4	0.67±0.17	3.25±1.89	0±0	0±0	0.50±0.58	0±0	77.010	1.143	4	0.53±0.03	3.00±0.82	0.75±0.50	0±0	0±0	0±0
	5	2.008	0.555	4	0.62±0.11	3.50±1.00	0.50±0.58	0±0	0.50±0.58	0±0	131.711	2.089	5	0.53±0.03	2.80±1.10	0.40±0.55	0±0	0±0	0±0

Table 8: Behavioral structure of the frequent patterns extracted with thresholds over 20%, 35% and 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 tasks, 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.

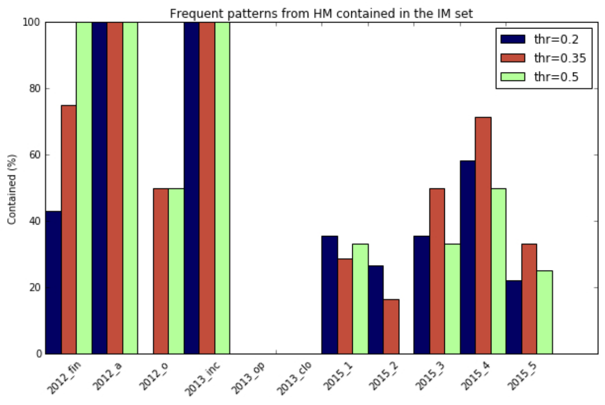
Regarding the runtime, we can observe that most of the values are close to 5 seconds, with the exception of the more complex models (*BPIC 2011*, *BPIC 2012-fin*, *BPIC 2013-inc*, IM of all *BPIC 2015*) ranging from 2 to 7 minutes. Most of this time is spent on the preprocessing, which can be shared 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.

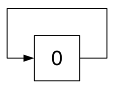


(a) Frequent patterns from the IM model contained in the frequent patterns set from the HM model.

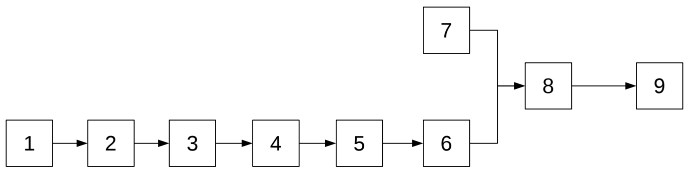


(b) Frequent patterns from the HM model contained in the frequent patterns set from the IM model.

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 task is *'aanname laboratoriumonderzoek'* (assumption laboratory).



(b) Frequent pattern (20.20%) 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 tasks 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.

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 tasks 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 task, 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 task, 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 20% 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.

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.

Acknowledgments.

This work was supported by the Spanish Ministry of Economy and Competitiveness (grant TIN2014-56633-C3-1-R co-funded by the European Regional Development Fund - FEDER program); the Galician Ministry of Education (projects EM2014/012, CN2012/151 and GRC2014/030); the Consellería de Cultura, Educación e Ordenación Universitaria (accreditation 2016-2019, ED431G/08); and the European Regional Development Fund (ERDF).

References

[1] Agrawal, R., Imieliński, T., Swami, A., 1993. Mining association rules between sets of items in large databases. In: Acm sigmod record. Vol. 22. ACM, pp. 207–216.

[2] Agrawal, R., Srikant, R., 1995. Mining sequential patterns. In: Data Engineering, 1995. Proceedings of the Eleventh International Conference on. 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: International Conference on Intelligent Data Engineering and Automated Learning. Springer, pp. 423–434.

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

[5] De San Pedro, J., Carmona, J., Cortadella, J., 2015. Log-based simplification of process models. In: International Conference on Business Process Management. Springer, pp. 457–474.

[6] Desel, J., Reisig, W., 1998. Place/transition petri nets. In: Lectures on Petri Nets I: Basic Models. Springer, pp. 122–173.

[7] Fahland, D., Van Der Aalst, W. M., 2011. Simplifying mined process models: An approach based on unfoldings. In: International Conference on Business Process Management. Springer, pp. 362–378.

[8] Greco, G., Guzzo, A., Manco, G., Pontieri, L., Saccà, D., 2006. Mining constrained graphs: The case of workflow systems. In: Constraint-Based Mining and Inductive Databases. Springer, pp. 155–171.

[9] Greco, G., Guzzo, A., Pontieri, L., Sacca, D., 2004. Mining expressive process models by clustering workflow traces. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, pp. 52–62.

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

[11] Günther, C. W., Rozinat, A., 2012. Disco: Discover your processes. BPM (Demos) 940, 40–44.

[12] Han, J., Cheng, H., Xin, D., Yan, X., 2007. Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery 15 (1), 55–86.

[13] Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.-C., 2000. Freespan: frequent pattern-projected sequential pattern mining. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp. 355–359.

[14] Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M., 2001. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: proceedings of the 17th international conference on data engineering. pp. 215–224.

[15] Leemans, M., van der Aalst, W. M., 2014. Discovery of frequent episodes in event logs. In: International Symposium on Data-Driven Process Discovery and Analysis. Springer, pp. 1–31.

[16] Leemans, S. J., Fahland, D., van der Aalst, W. M., 2013. Discovering block-structured process models from event logs-a constructive approach. In: International Conference on Applications and Theory of Petri Nets and Concurrency. Springer, pp. 311–329.

[17] Mannila, H., Toivonen, H., Verkamo, A. I., 1997. Discovery of frequent episodes in event sequences. Data mining and knowledge discovery 1 (3), 259–289.

[18] Song, M., Günther, C. W., Van der Aalst, W. M., 2008. Trace clustering in process mining. In: International Conference on Business Process Management. Springer, pp. 109–120.

[19] Steeman, W., 2013. Bpi challenge 2013, closed problems.
URL `https://doi.org/10.4121/uuid:c2c3b154-ab26-4b31-a0e8-8f2350ddac11`

[20] Steeman, W., 2013. Bpi challenge 2013, incidents.
URL `https://doi.org/10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee`

[21] Steeman, W., 2013. Bpi challenge 2013, open problems.
URL `https://doi.org/10.4121/uuid:3537c19d-6c64-4b1d-815d-915ab0e479da`

[22] Tax, N., Sidorova, N., Haakma, R., van der Aalst, W. M., 2016. Mining local process models. Journal of Innovation in Digital Ecosystems.

[23] van der Aalst, W., 2011. Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st Edition. Springer.

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

[25] van Dongen, B., 2012. Bpi challenge 2012.
URL `https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`

[26] van Dongen, B., 2015. Bpi challenge 2015.
URL `https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1`

[27] Van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H., Weijters, A., Van Der Aalst, W. M., 2005. The prom framework: A new era in process mining tool support. In: International Conference on Application and Theory of Petri Nets. Springer, pp. 444–454.

[28] Vázquez-Barreiros, B., Lama, M., Mucientes, M., Vidal, J. C., 2014. Softlearn: A process mining platform for the discovery of learning paths. In: 2014 IEEE 14th International Conference on Advanced Learning Technologies. IEEE, pp. 373–375.

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

[30] Weijters, A., van Der Aalst, W. M., De Medeiros, A. A., 2006. Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP 166, 1–34.

[31] Zaki, M. J., 2001. Spade: An efficient algorithm for mining frequent sequences. Machine learning 42 (1-2), 31–60.