Towards the Extraction of Frequent Patterns in Complex Process Models

David Chapela-Campa, Manuel Mucientes, and Manuel Lama

Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)
Universidade de Santiago de Compostela. Santiago de Compostela, Spain
{david.chapela, manuel.mucientes, manuel.lama}@usc.es

Abstract. 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, 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

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* [1]. 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.

To discover a complex process model, i.e., a hardly readable process model, can totally hinder its quality [2] making difficult the retrieval of behavioural information. Different techniques have been proposed to tackle this problem: the simplification of already mined models [2,3], the search of simpler structures in the logs [4,5,6], or the clusterization of the log into smaller and more homogeneous subsets of traces to discover different models within the same process [7,8]. 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, in this paper, we will focus in the search of frequent behavioural patterns based on the repetition of this patterns in the traces of the log. The extraction of these frequent structures is useful in highly complex models —it allows to abstract from all the behaviour and focus on relevant structures— and well-structured models —it retrieves frequent subprocesses which can be, for instance, the objective of optimizations.

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 [9].

It provides a simple technique to 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, can build frequent patterns based just on the logs, searching in them for frequent sequences of tasks [10,11]. Improving this search, episode mining techniques focus their search in frequent, and more complex structures such as parallels [4,5]. With a different approach, the *w*-find algorithm [12] uses the process model to build the patterns, checking their frequency in the logs. Finally, extending these mining techniques, the local process mining approach of Niek Tax et al. [6] discovers frequent patterns from the logs providing support to loops. Nevertheless, all these techniques fail to measure the frequency of a pattern in some cases, and specially when the model presents loops or optional tasks[1].

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 [12], 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 several synthetic process models —containing loops, parallelisms, selections and sequences—, and with 12 real complex logs of the Business Process Intelligence Challenges.

2 Preliminaries

Definition 1 (Causal matrix). A Causal matrix is formed by the set of tasks T of the model it represents. And two sets for each of the tasks with the inputs and outputs of it. 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. $O(\alpha)$ contains the same information for the outputs.

Definition 2 (Pattern). A *pattern* 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 an incomplete parallel connection.

Definition 3 (Simple Pattern). A *simple pattern* is a pattern whose 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

[1] 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.

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.

Definition 4 (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 its closure, 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.

Definition 5 (Candidate Arcs). The set of candidate arcs, or $A^<$, is a subset of the arcs in the model which are not part of an AND structure.

Definition 6 (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.

Definition 7 (Frequecy of a Pattern). The frequency of a simple pattern SP is the number of traces of the log compliant with SP, divided by the size of the log.

And, the frequency of a pattern P is the minimum frequency of the simple patterns it contains.

Definition 8 (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 $freq(P) \geq \sigma$.

3 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. Instead of a brute-force technique, WoMine performs an a priori search [12] starting with the frequent minimal patterns (Def. 4) of the model. This search includes an expansion stage done in two ways: *i)* adding frequent M-patterns not contained in the current pattern, and *ii)* adding frequent arcs of the $A^<$ set (Def. 5). This expansion is followed by a pruning strategy that verifies the downward-closure property of support [13] — 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 part of a frequent pattern.

The pseudocode in Alg. 1 shows the main structure of WoMine. First, the frequent arcs of $A^<$ and the frequent M-patterns are initialized, using Alg. 2 to measure the frequency. 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).

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.

```
1  Algorithm WoMine(W, T, thr)
2      M ← {m | m ∈ W, m is an M-pattern } // Def. 4
3      A< ← {a | a ∈ W, a is a Candidate Arc } // Def. 5
4      frequentArcs ← {a | a ∈ A<, a is frequent w.r.t. T}
5      frequentM ← {m | m ∈ M, isFrequentPattern(m, T, thr) } // using Alg. 2
6      frequentPatterns ← frequentM
7      currentPatterns ← frequentM
8      while currentPatterns ≠ ∅ do
9          candPatterns ← ∅
10         forall the p ∈ currentPatterns do
11             candPatterns ← candPatterns ∪ addFrequentArcs(p)
12             complementaryM ← {m | m ∈ M, m ∉ p}
13             forall the m ∈ complementaryM do
14                 candPatterns ← candPatterns ∪ addFrequentMPattern(p, m)
15             end
16         end
17         currentPatterns ← {p | p ∈ candPatterns, isFrequentPattern(p, T, thr)}
               // using Alg. 2
18         frequentPatterns ← frequentPatterns ∪ currentPatterns
19     end
20     Delete the redundant patterns of frequentPatterns
21     return frequentPatterns
```

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`. 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`. Once the expansion is completed, the obtained patterns are filtered to delete the infrequent ones (Alg. 1:17). Finally, once the iterative stage finishes, a simplification is made to delete the patterns which provide redundant information (Alg. 1:20). This simplification stage consists in the deletion of the patterns whose behaviour is contained inside others.

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.

4 Measuring the Frequency of a Pattern

In each step of the iterative stage, 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. 7 and 8, 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

Algorithm 2. Check if a pattern is frequent.

Input: A set $T = T_1, ..., T_N$ of traces, a pattern *pattern* to measure its frequency w.r.t. T and a threshold to establish the bound of frequency.
Output: A Boolean value indicating if the pattern is frequent or not.

```
1  Algorithm isFrequentPattern(pattern, T, threshold)
2      simplePatterns ← generate the simple patterns of pattern
3      frequencies ← ∅
4      forall the simplePattern ∈ simplePatterns do
5          frequencies ← frequencies ∪ getPatternFrequency(simplePattern,
           T)
6      end
7      minFreq ← 0
8      if frequencies.length > 0 then
9          minFreq ← minimum of frequencies
10     end
11     realFreq ← minFreq/T.length
12     return realFreq ≥ threshold
13 Function getPatternFrequency(pattern, T)
14     executed ← 0
15     forall the trace ∈ T do
16         if isTraceCompliant(pattern, trace) then
17             executed ← executed + 1
18         end
19     end
20     return executed
21 Function isTraceCompliant(pattern, trace)
22     forall the task ∈ trace do
23         Execute task in the process model
24         sources ← get the tasks that fired the execution of task
25         simulateExecutionInPattern(sources, task, pattern)
26         if pattern has been successfully executed then
27             return true
28         end
29     end
30     return false
```

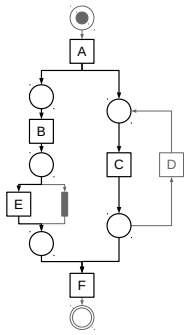
(`simulateExecutionInPattern`) consists in a replay of the trace, checking if the pattern is executed correctly.

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. 1 shows an example of this stage. The algorithm starts (#0) with the sets of the *executed arcs* and *last executed tasks* empty. The first step (#1) executes `A`. There are 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 step, task `E` (#3), has the same behaviour. The executed arc is in the pattern and its source task is in the *last executed tasks* set. Hence, the *executed arcs* set is updated and `B` replaced by `E` in the *last executed tasks* set. After this stage, the following task is `C` (#4). 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.

Finally (#5), `F` has two firing tasks and, thus, two arcs are executed. In both cases, the source task of the arcs —`C` and `E`— is in the *last executed tasks* set, and the arc is in the pattern. Thus, a simple addition of `F` to the *last executed tasks* set is done when the last of its branches is executed.



(a) Petri net of a process model with a pattern highlighted in black (the unnamed task is an invisible task).

Trace: A B E C F			
Initial tasks: {A}			
End tasks: {F}			
#	**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	C	$\langle A \rightarrow B \rangle, \langle B \rightarrow E \rangle, \langle A \rightarrow C \rangle$	E, C
5	F	$\langle A \rightarrow B \rangle, \langle B \rightarrow E \rangle, \langle A \rightarrow C \rangle, \langle C \rightarrow F \rangle, \langle E \rightarrow F \rangle$	F

(b) Check of the execution of a trace for the pattern highlighted in Fig. 1a: '#' 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 whose execution was correctly saved; *'Last executed tasks'* is the set of tasks which have not fired an entire set of their outputs.

Fig. 1: An example that shows how the algorithm checks if a trace is compliant with a pattern of the process model.

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 `F` is executed. Thus, the trace is compliant with the pattern.

The stage 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 #5, if the arc $\langle C \rightarrow D \rangle$ was executed, this would cause this saving stage 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. This analysis is able to recognize the correct execution of a pattern in 1-safe Petri nets[2].

5 Experimentation

The validation of the presented approach has been done with different types of event logs. A comparison between WoMine and the state of the art techniques has been done, using 5 process models with the most common control structures[3]. The results of these techniques over the models are presented in Table 1. As can be seen, WoMine is able to retrieve frequent patterns while the other techniques fail to do so in some cases. In Subsection 5.1, 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)[4].

[2] 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.

[3] Models available in `http://tec.citius.usc.es/processmining/womine/`

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

	Examples				
	Ex. #1	**Ex. #2**	**Ex. #3**	**Ex. #4**	**Ex. #5**
WoMine	+	+	+	+	+
Heat Maps [9]	±	-	-	+	-
w-find [12]	+	±	-	-	-
Local Process Mining [6]	+	±	±	-	±
Episode Mining [4]	+	±	-	-	-
SPM (PrefixSpan) [11]	+	-	±	-	-

Table 1: 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 (wrong frequency or wrong structure).

5.1 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) [14] and, on the other hand, to analyze the influence of the model in the retrieved patterns. Some BPIC logs [15,16,17,14,18,19] have been used for this purpose. These logs have been mined with two of the most popular discovery algorithms, the Heuristics Miner (HM) [20] and the Inductive Miner (IM) [21].

Table 2 shows the results of WoMine over these models for a threshold of 20%. The results demonstrate the ability of WoMine to extract patterns with loops, choices, parallels and sequences. Regarding the runtime, we can observe that most of the values are close to 5 seconds, with the exception of the more complex models (*2011*, *2012-fin*, *2013-inc*, IM of all *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

		Threshold : 20%								
		Heuristics Miner								
		runtime (secs)		#patt	frequency	#tasks	#sequences	#choices	#parallels	#loops
		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
	a	0.014	0.029	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
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
	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
	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
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
	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
	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
	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
	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
		Inductive Miner								
2011		-	-	-	-	-	-	-	-	-
2012	fin	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	0.011	1	0.84±0.00	5.00±0.00	1.00±0.00	0±0	0±0	0±0
	o	0.077	0.008	2	0.75±0.35	2.00±0.00	0±0	0±0	0±0	0±0
2013	inc	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	0.082	0.017	2	0.76±0.34	1.50±0.71	0±0	0±0	0±0	0.50±0.71
	op	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	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	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	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	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	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

Table 2: Behavioral structure of the frequent patterns extracted for a threshold of 20% from the process models of the BPICs. It shows the information for the results with two process models of each log (Heuristics and Inductive). The information contains the runtime, the number of patterns and the distribution (average and standard deviation) of the frequency, the number of 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.

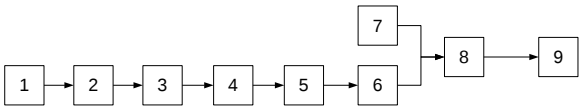


Fig. 2: Frequent pattern (20.20%) extracted from the 2011 log with the Heuristics model. The pattern is formed by two sequences joined by a choice (XOR-join).

applications. Also, there is a significant difference in the algorithm runtime between models. These differences are due to the very different grades of complexity of the mined models —IM models present far more relations than HM models.

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 implies, as seen before, a higher runtime. 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.

Fig. 2 shows a example of a pattern 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. The pattern, detected by WoMine in the 20% of the traces, is formed by two sequences, joined by a choice. With this information, the process manager may try to optimize the subprocess, or schedule the resources to improve the execution of the process.

6 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. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. 1st edn. Springer (2011)
2. De San Pedro, J., Carmona, J., Cortadella, J.: Log-based simplification of process models. In: International Conference on Business Process Management, Springer (2015) 457–474
3. Fahland, D., Van Der Aalst, W.M.: Simplifying mined process models: An approach based on unfoldings. In: International Conference on Business Process Management, Springer (2011) 362–378
4. Leemans, M., van der Aalst, W.M.: Discovery of frequent episodes in event logs. In: International Symposium on Data-Driven Process Discovery and Analysis, Springer (2014) 1–31
5. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data mining and knowledge discovery **1**(3) (1997) 259–289
6. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.: Mining local process models. Journal of Innovation in Digital Ecosystems (2016)
7. Greco, G., Guzzo, A., Pontieri, L., Sacca, D.: Discovering expressive process models by clustering log traces. IEEE Transactions on Knowledge and Data Engineering **18**(8) (2006) 1010–1027
8. Song, M., Günther, C.W., Van der Aalst, W.M.: Trace clustering in process mining. In: International Conference on Business Process Management, Springer (2008) 109–120
9. Günther, C.W., Rozinat, A.: Disco: Discover your processes. BPM (Demos) **940** (2012) 40–44
10. Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.C.: Freespan: frequent pattern-projected sequential pattern mining. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (2000) 355–359
11. Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: proceedings of the 17th international conference on data engineering. (2001) 215–224
12. Greco, G., Guzzo, A., Manco, G., Pontieri, L., Saccà, D.: Mining constrained graphs: The case of workflow systems. In: Constraint-Based Mining and Inductive Databases. Springer (2006) 155–171
13. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Acm sigmod record. Volume 22., ACM (1993) 207–216
14. van Dongen, B.: Real-life event logs - hospital log (2011)
15. Steeman, W.: Bpi challenge 2013, closed problems (2013)
16. Steeman, W.: Bpi challenge 2013, incidents (2013)
17. Steeman, W.: Bpi challenge 2013, open problems (2013)
18. van Dongen, B.: Bpi challenge 2012 (2012)
19. van Dongen, B.: Bpi challenge 2015 (2015)
20. Weijters, A., van Der Aalst, W.M., De Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP **166** (2006) 1–34
21. Leemans, S.J., Fahland, D., van der Aalst, W.M.: Discovering block-structured process models from event logs-a constructive approach. In: International Conference on Applications and Theory of Petri Nets and Concurrency, Springer (2013) 311–329