# A Process Mining Technique Using Pattern Recognition

Veronica Liesaputra[1], Sira Yongchareon[1], and Sivadon Chaisiri[2]

[1]Department of Computing and Information Technology
Unitec Institute of Technology, New Zealand
vliesaputra@unitec.ac.nz, sira@maxsira.com

[2]School of Information Technology
Shinawatra University, Thailand
sivadon@ieee.org

**Abstract.** Several works have proposed process mining techniques to discover process models from event logs. With the existing works, mined models can be built based on analyzing the relationship between any two events seen in event logs. Being restricted by that, they can only handle special cases of routing constructs and often produce unsound models that do not cover all of the traces in the logs. In this paper, we propose a novel technique for process mining based on using a pattern recognition technique called *Maximal Pattern Mining* (MPM). Our MPM technique can handle loops (of any length), duplicate tasks, non-free choice constructs, and long distance dependencies. Furthermore, by using the MPM, the discovered models are generally much easier to understand.

## 1. Introduction

Since the mid-nineties, several techniques have been proposed to automatically discover process models from event logs in both software processes and business process domains [2, 3, 4]. Several algorithms are variants of the $\alpha$-algorithm (e.g., in [5, 6, 7, 8]), which is regarded as a well-known technique for process discovery that pioneered studies in this field. Nevertheless, due to the fact that the $\alpha$-algorithms face problems dealing with complicated routing constructs, noise, and incompletes [1], more advanced techniques, such as region-based approaches (e.g., [14, 15, 16, 18]), heuristic mining [9], fuzzy mining [10], and genetic mining [11], have been proposed to tackle these aforementioned problems.

We argue that the existing algorithms for discovering process models are still unable to efficiently and accurately handle loops (of any length), duplicate tasks, concurrency, long dependencies and complex routing constructs. In fact, some of these algorithms may produce unsound models. To address these problems, we propose a novel process mining technique called *Maximal Pattern Mining* (MPM). Instead of mining the relationship between two events, MPM mines a set of patterns that could cover all of the traces seen in an event log. The time needed by our algorithm to process mine and generate a process model is also significantly shorter than all the existing algorithms.

The remainder of the paper is organized as follows. Section 2 reviews and discusses the work that has been done in the process mining area. Section 3 proposes our MPM

technique for process discovery. Section 4 discusses our preliminary evaluation. Finally, the conclusion and future works are given in Section 5.

## 2. Background and Related Work

Van der Aalst et al. [5] proposed $\alpha$-algorithm to discover structured workflow nets from complete event logs. However, the $\alpha$-algorithm cannot cope with noise, incompleteness of workflow logs, short loops, and non-free choice constructs. Later, Alves de Medeiros et al. [6] developed $\alpha^+$-algorithm, an improved version of $\alpha$-algorithm, which is capable of detecting short loops. Further, Wen et al. [7, 8] proposed $\alpha^{++}$-algorithm to discover non-free choice constructs and $\beta$-algorithm to detect concurrency. Due to the fact that all $\alpha$-algorithms face the same robustness problem, Weijters et al. [9] proposed Heuristics Miner by extending the $\alpha$-algorithm to analyze the frequency of the three types of relationships between activities in a workflow log: direct dependency, concurrency, and not-directly connectedness. In contrast to the $\alpha$-algorithms, Gunther and van der Aalst [10] proposed Fuzzy Miner, an adaptive technique to discover behavior models from an event log using significance and correlation measures.

Van der Werf et al. [16] proposed a discovery technique using Integer Linear Programming (ILP) based on the theory of regions. Van der Aalst et al. [14] proposed a Finite State Machine (FSM) Miner/Petrify two-step approach to find a balanced trade-off between generalization and precision of discovered process models. The theory of region is used in their approach as a method to bridge FSM and Petri-Net models as also proposed in [15]. Sole and Carmona [18] presented an aggressive folding region-based technique, which is based on the theory of region, to reduce the total number of states of a transition system and speed up the discovery process. Alves de Medeiros et al. [11] proposed a genetic algorithm which performs a global search based on the use of fitness function using both a recall and a precision measure to find the best matched models. The DT Genetic and Genetic Miner can detect non-local patterns and, due to its post-pruning step, it has a reasonable robustness. While the latter cannot detect duplicate tasks, the former can detect them. Similarly, Goedertier et al. [12] proposed AGNEsMiner to deal with problems such as expressiveness, noise, incomplete event logs, and the inclusion of prior knowledge by representing process discovery as a multi-relational classification problem [13] on event logs supplemented with Artificially Generated Negative Events (AGNEs). This technique can learn the conditions that distinguish between the occurrence of either a positive or a negative event.

Based on the above discussion, we have observed that only the DT Genetic Miner [11] can tackle all of the typical process mining problems, i.e., noise, duplicate tasks, hidden tasks, non-free choice constructs, and loops. However, because of the nature of the genetic algorithm, it consumes much more processing time and space in order to learn and construct a model. Mining efficiency is considered a major drawback of this approach in which it is undesirable, especially when it is applied to a complicated real-life log. To overcome such issues, we need to develop a better technique that not only solves all the typical process mining problems but also requires far less processing time.

## 3. Maximal Pattern Mining (MPM)

Instead of looking at the relationship between two events which is what most of the existing process mining techniques focus on, we propose a pattern mining technique to analyse the whole sequence of events in all of the traces and find the optimal set of "regular expression"-like patterns that would cover them. Our MPM technique is described in Sections 3.1. Assumptions and limitations of the technique are discussed in Section 3.2.

### 3.1. Overview

Let $T = \{t_0, t_1 \dots t_n\}$ be the collections of all the traces in an event log that is ordered first by the value of the events in the trace and then by the number of events in the trace. A trace $t_n$ is an ordered sequence of events or completed tasks, $t_n = \langle z_0, z_1 \dots z_m \rangle$. We denote $|t_n|$ as the number of events in a trace. An event $z_m$ only contains 1 event type, i.e. $|z_m| = 1$. All the traces and events in $T$ and $t_n$ are not unique, i.e. it is possible to have $T = \{\langle a,b,c,b,b,c,d,e\rangle,\langle a,b,c,b,b,c,d,e\rangle,\langle a,b,b,c,e,d\rangle\}$. Given an input $T$, our algorithm will first create a list of unique patterns $P = \{p_0, p_1 \dots p_i\}$ and then generate a graph based on $P$. The following sections will describe each of them. A pattern $p_i = \langle e_0, e_1 \dots e_j\rangle$ is an ordered sequence of elements, $|p_i|$ is the number of elements in the pattern and $p_i.support$ is the number of traces covered by the pattern. An element $e_j = \{v_0, v_1 \dots v_k\}$ contains $k$ number of unique event types (i.e. $|e_j| = k$) and $e_j.loop$ is a list of $\langle v_k: w\rangle$ tuples that indicate whether $v_k$ is self-looping ($w = \{v_k\}$) and/or is the last element of a sequence-loop ($w = \{e_x\ e_{x+1} \dots e_{x+y}\}$ and $e_{x+y} = v_k$). The *loop* list is ordered first by the event value and then by the number of elements in $w$ ($|w|$). An element's value $v_k$ only contains 1 event type. All the elements inside $p_i$ might not be unique. For instance, given the $T = \{\langle a,b,c,b,b,c,d,e\rangle, \langle a,b,c,b,b,c,d,e\rangle, \langle a,b,b,c,e,d\rangle\}$ specified above, our algorithm will only produce 1 pattern in $P$. $p_0 = \langle e_0, e_1, e_2, e_3\rangle$, where $e_0 = a$ and $e_0.loop = \emptyset$; $e_1 = b$ and $e_1.loop = \emptyset$; $e_2 = c$ and $e_2.loop = \{\langle c: \{bc\}\rangle\}$; and $e_3 = \{d, e\}$ and $e_3.loop = \emptyset$. Elements with more than one event type indicate a parallelization. In our example, $e_3$ shows that in the last 2 events of our model the values could be either *de* or *ed*. Because $p_0$ covers all the traces in $T$, $p_0.support = 3$.

Our graph algorithm will then generate the following model (Fig. 1) based on $p_0$. We use the operator AND to indicate the set of tasks that are running at the same time, and XOR to indicate a path selection.

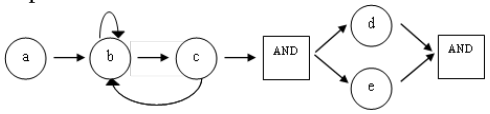



**Figure 1.** The generated model for $\{\langle a,b,c,b,b,c,d,e\rangle,\langle a,b,c,b,b,c,d,e\rangle,\langle a,b,b,c,e,d\rangle\}$

The algorithm we use to construct the most optimal patterns for a given trace of events has five main phases: finding self and/or sequence loops, storing the pattern in a vertical format, identifying events that should be done concurrently, investigating whether a trace is covered by a pattern in $P$, and pruning non-maximal patterns.

**Loops.** A sequence of elements $S = \langle s_0, s_1 \dots s_q\rangle$ is in a loop in the trace $t_n = \langle z_0, z_1 \dots z_m\rangle$ or in the pattern $p_i = \langle e_0, e_1 \dots e_m\rangle$ if and only if there is a sequence of elements such

that for all $b \in \{0 \ldots q\}$and $q \leq (m - a)/2$, $z_{a+b} = s_b$ and $z_{a+q+b} = s_b$ or $e_{a+b} = s_b$ and $e_{a+q+b} = s_b$, where $a$ is the starting index where $S$ occurs in the trace or in the pattern ($0 \leq a \leq m$). The first phase of our pattern mining is to identify these loops. For every $S+$ occurring in $t_n$ and $p_i$, we replace it with $S$ and set the *loop* property of the last element in $S$. For instance, given a pattern ⟨*a,b,b,c,d,{e,f},c,d,{e,f}c,d,{e,f}g*⟩, the pattern becomes ⟨*a,b,c,d,{e,f}g*⟩ where the loop property for $b$ is $b$, and the loop property for *{e,f}* is *cd{e,f}*. By identifying loops first, MPM would be able to deduce that traces ⟨*a,b,d,d,c,b,b,b,d,c,b,d,c,e*⟩ and ⟨*a,b,d,c,b,d,d,c,e*⟩ are the same and are both covered by the pattern ⟨*a,b,d,c,e*⟩.

**Vertical Representation.** Existing process mining algorithms require several scans of the event logs or need to maintain large amounts of intermediate candidates in the main memory to generate process models [7, 8, 11, 13]. To alleviate this problem, MPM stores all patterns in the vertical format as an IdList in bitset representation [20] where each entry represents an element with the id of the trace where the element appears (*id*) and the position (*pos*) where it appears. The support of a pattern is calculated by making joint operations with IdLists of smaller patterns. Thus, MPM would only need to perform a single scan through the log to generate an IdList of patterns containing single elements (see [20] for details). To make it more verbose, MPM uses the symbol $ to indicate the end of a trace. Given $T = \{\langle a,b,c,b,b,c,d,e,a\rangle, \langle a,b,b,c,e,d,a\rangle, \langle e,d,a\rangle\}$, the vertical representation ($V_T$) of it is represented as follows:

| A | |
|---|---|
| **id** | **pos** |
| 0 | 0, 5 |
| 1 | 0, 5 |
| 2 | 2 |

| b | |
|---|---|
| **id** | **pos** |
| 0 | 1 |
| 1 | 1 |

| c | |
|---|---|
| **id** | **pos** |
| 0 | 2 |
| 1 | 2 |

| d | |
|---|---|
| **id** | **pos** |
| 0 | 3 |
| 1 | 4 |
| 2 | 1 |

| e | |
|---|---|
| **id** | **pos** |
| 0 | 4 |
| 1 | 3 |
| 2 | 0 |

| $ | |
|---|---|
| **id** | **pos** |
| 0 | 6 |
| 1 | 6 |
| 2 | 3 |

**Concurrency.** The next phase of our pattern mining is to identify tasks that should be done in parallel. A set of events $V = \{v_0, v_1 \ldots v_q\}$ are performed at the same time if and only if there are at least $q$ number of unique traces with the following sequence $\langle z_0, z_1 \ldots z_{a-1}\ z_a, z_{a+1} \ldots z_{a+q}\ z_{a+q+1}, z_{a+q+2} \ldots z_m\rangle$, where the sequence $\langle z_0, z_1 \ldots z_{a-1}\rangle$ and $\langle z_{a+q+1}, z_{a+q+2} \ldots z_m\rangle$ have the same pattern across those traces, there are no events mentioned more than once in $\langle z_a, z_{a+1} \ldots z_{a+q}\rangle$, and for all $b \in \{0 \ldots q\}$and $q \leq (m - a)$, $z_{a+b} \subseteq V$, where $a$ is the starting index where a combination of all the events in $V$ occur ($0 \leq a \leq m$). Sequence $\langle z_0, z_1 \ldots z_{a-1}\rangle$ and $\langle z_{a+q+1}, z_{a+q+2} \ldots z_m\rangle$ may be $\emptyset$. Instead of $z_{a+b} = V$, we relax the criteria to $z_{a+b} \subseteq V$ with the assumption that if we see almost all of $V$ possible events combined in $T$, it must be that the trace log is incomplete. For example, given a set of traces *{⟨a,b,c,d,e⟩,⟨a,b,d,c,e⟩,⟨a,c,d,b,e⟩}*, we first look at the first two traces where we get ⟨*a,b,{c,d},e*⟩ as it is possible to switch the position of task $c$ and $d$ around. We then compare it with the last trace where we get ⟨*a,{b,c,d},e*⟩ as we can switch the position of task $c$ and $d$ around with $b$. In the future, we may use the trace frequency to help us decide when we should use the strict or relaxed criteria.

**Coverage.** A pattern $p_i = \langle e_0, e_1 \ldots e_n\rangle$ specifies the sequence of patterns that covers some of the traces in $T$ and it can be represented as a deterministic finite automata $DFA_i$ with (a) a well-defined start state, (b) one or more accepted states and (c) deterministic transitions across states on symbols of the event values. A trace $t_n = \langle z_0, z_1 \ldots z_m\rangle$ is covered by the pattern $p_i$ if and only if the sequence of transitions for the elements of $t_n$

from the start state results in an accepted state. Fig. 2 illustrates the deterministic finite automaton for the pattern ⟨*a,b,{c,d},e*⟩ with the *loop* property for *b* to be *b*. We use > to indicate the start state and double circles for the accept state. The diagram shows that the pattern covers the following set of traces *{⟨a, b, c, d, e⟩, ⟨a, b, d, c, e⟩, ⟨a, b, b, c, d, e⟩, ⟨a, b,…,b, c, d, e⟩, ⟨a, b,…,b, d, c, e⟩}*. However, it will reject the following set of traces *{⟨a,b⟩, ⟨e⟩, ⟨a,b,h⟩, ⟨a,b,c,d⟩, ⟨a,b,b,d,c⟩, ⟨a,b,a,d,c,e⟩}*. Because we have to go through each of the elements of $t_n$ to identify events that should be done in parallel, we perform both tasks simultaneously.

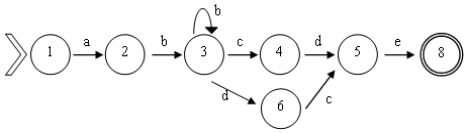



**Figure 2.** The deterministic finite automata model for $p_i$ = ⟨*a,b,{c,d},e*⟩

**Maximal Patterns.** A pattern $p_i$ is said to be maximal if and only if there is no other pattern $p_j$ in *P* that has the same start and accept states and covers the same or more traces in *T*. Given *P = {⟨a,b,c,d⟩,⟨a,{b,c},d⟩,⟨a,b,c⟩}*, only $p_1$ and $p_2$ are maximal because $p_0$ is a sub-pattern of $p_1$.

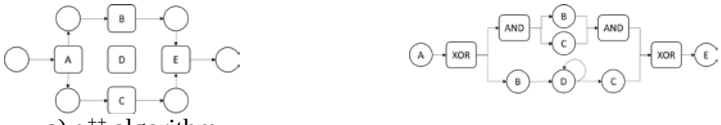
**Noise.** To further filter *P* from noisy data, we set a support threshold value, *thresh*, such that we would only keep frequent patterns $p_i$ and events $v_k$, i.e. $p_i.support \geq thresh$ and $v_k.support \geq thresh$. All patterns and events are accepted if the threshold value is 0.

### 3.2. Assumptions and Limitations

An event in a transactional log usually contains information such as the event type/value (e.g. apply for a drivers licence or update a patients information), the agent/performer that initiates the event, timestamp and the data element being modified or accessed (e.g. the age of a patient, the driving test result). Because the goal of MPM is to find all possible orderings of the logged events in the system, only the event's type or value are mined. Other information, such as the timestamp and agent, are removed from the logs. In our setting, we know the original model that our algorithm should strive to construct, the complete list of traces that the model can generate, and the instances in a log that are negative examples. But in real life scenarios, no original model is available. Logs may contain noise such as mislabelled events, incorrectly logged sequences of events and exceptions. In fact, a particular trace of events observed does not mean that the model should be able to reproduce it. Furthermore, in a complex process with many possible paths, only a fraction of those paths may be present in the log, i.e., the log is incomplete. Thus, it is undesirable to construct a model that allows only for the observed instances in the log. Since we do not know which instance in the log is noise, we assume that every trace/event that is recorded in the log and appears no less than a user's specified threshold frequency is correct (positive examples). However, unobserved traces of events are not considered as negative examples. Our MPM algorithm can construct a model that can explain all the traces of events found in the logs while also allowing for any unobserved behaviour.

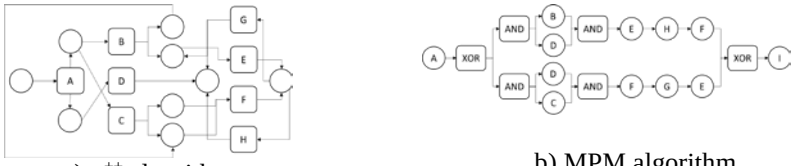
## 4. Preliminary evaluation

We evaluated the quality of the mined model produced by MPM, $\alpha^{++}$, DT genetic miner, AGNEs and heuristic miners according to logs that are mentioned in their respective publications. We did not perform the evaluation on $\alpha$ and $\alpha^{+}$ as [7, 8] have reported that $\alpha^{++}$ can construct a model that handles more complex control-flow constructs. Similar to other discovery algorithms, our MPM algorithm is implemented as a plugin of ProM [19]. In our initial evaluation, we use synthetic log data to demonstrate the fact that the MPM algorithm can significantly improve the performance of the existing approaches, especially the $\alpha$-algorithm and its variants. We do not use parameter fine-tuning or metadata to enhance the performance of our algorithm. We have also used the default settings for $\alpha^{++}$, genetic miner and AGNEs. To further extend the capability of Heuristic Miners, we configure it to discover long distance dependencies based on completed events' values and positions on a trace. Due to the fact that the $\alpha^{++}$ algorithm builds a process model based on the relationship between *any two* events so that it does not allow an event to occur more than once in the model, it requires additional heuristics to handle long distance dependency, short loops (maximum of two events) and non-free-choice constructs (combination of choice and concurrency); and assumes that two or more events must occur concurrently if they have the same parents (i.e. low precision). Therefore, it is possible for the $\alpha^{++}$ algorithm to produce *unsound* workflow nets as shown Figures 3 and 4. Similarly, because Heuristic Miners also builds a casual matrix that represents the relationship between any two events, it cannot handle duplicate tasks as illustrated in Figure 5. Although AGNEs is more versatile than Heuristic Miners, it is still incapable of handling a complex non-free choice construct such as is displayed in Figure 6.



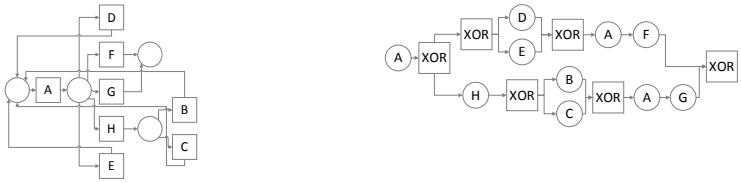

a) $\alpha^{++}$ algorithm b) MPM algorithm

**Figure 3.** Log *T* = {*ABCE, ACBE, ABDDCE*}



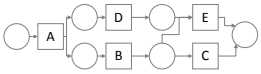

a) $\alpha^{++}$ algorithm b) MPM algorithm

**Figure 4.** Log *T* = {*ABDEHFI, ADBEHFI, ACDFGEI, ADCFGEI*}

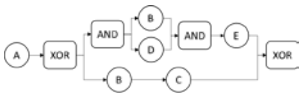



a) Heuristic Miner b) MPM algorithm

**Figure 5.** Log *T* = {*ADAF, AEAF, AHBAG, AHCAG*}



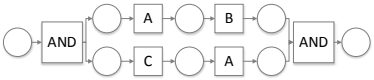

a)AGNES



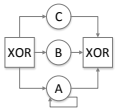

b) MPM algorithm

**Figure 6.** Log *T* = {*ABC, ABDE, ADBE*}

Our MPM algorithm discovers a process model by reading patterns from the *whole sequence* of events in the traces. Thus, its criteria is more stringent than Heuristic Miners or $\alpha^{++}$; it can handle duplicate tasks, long distance dependencies, loops of any length and non-free choice constructs. The process model discovered by MPM is always sound, and it is generally more accurate and readable than the models mined by AGNEs, Heuristic Miners or $\alpha^{++}$. However, MPM is incapable of generating a model that accurately represents duplicate tasks in a parallel process structure, as shown in Figure 7. DT Genetic Algorithm is the only algorithm that can correctly mine this log.




a) DT Genetic Miner




b) MPM algorithm

**Figure 7.** Log *T* = {*ABC, ABDE, ADBE*}

While DT Genetic Miner will sometimes produce a model that is more accurate than MPM, MPM can generate a similar model in significantly less time. Furthermore, MPM can build and improve the mined model incrementally in near real time as it receives new traces of events, i.e. the model becomes more accurate as it sees more unique traces of events.

## 5. Conclusion and Future work

In this paper, we propose a novel technique called *Maximum Pattern Mining* (MPM) to discover a process model from event logs. We have implemented our technique with preliminary evaluations against well-known process discovery algorithms: $\alpha^{++}$, DT genetic miner, AGNEs and the Heuristic Miners algorithm. Our results show that it can handle more general cases, such as loops of any length and long distance dependencies. In the future, we will implement and improve this technique with evaluations on real-life logs to see if our algorithm can handle very complex and very large logs.

## References

1. van der Aalst, W.M.P.: Process Mining: Overview and Opportunities, *ACM Transactions on Management Information Systems*, 2012, vol. 3, no. 2, article 7.

2. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs, in: *Proceedings of the 6th International Conference on Extending Database Technology* (EDBT'98), 1998, LNCS 1377, pp. 469-483.
3. Cook, J., Wolf, A.: Discovering models of software processes from event-based data, *ACM Transactions on Software Engineering and Methodology*, 1998 (7), pp. 215–249.
4. Datta, A.: Automating the discovery of AS-IS business process models: probabilistic and algorithmic approaches, *Information Systems Research*, 1998, vol. 9, pp. 275–301.
5. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: discovering process models from event logs, *IEEE Transactions on Knowledge and Data Engineering*, 2004, vol. 16, pp. 1128–1142.
6. Alves de Medeiros, A.K., van Dongen, B.F., van der Aalst, W.M.P., Weijters, A.J.M.M.: Process Mining: Extending the Alpha-Algorithm to Mine Short Loops, *BETA Working Paper Series*, TU Eindho- ven, 2004, vol. 113.
7. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs, *Data Mining and Knowledge Discovery*, 2007 (15), pp. 145–180.
8. Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., Sun, J.: A novel approach for process mining based on event types, *Journal of Intelligent Information Systems*, 2009, vol. 32, pp. 163–190.
9. Weijters, A.J.M.M., van der Aalst, W.M.P., Alves de Medeiros, A.K.: Process Mining with the Heuristics Miner algorithm, *BETA Working Paper Series*, 2006, TU Eindhoven, vol. 166.
10. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining - adaptive process simplification based on multi-perspective metrics, in: *Proceedings of the 5th International Conference on Business Process Management* (BPM), 2007, LNCS 4714, pp. 328–343.
11. Alves de Medeiros, A.K., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic process mining: an experimental evaluation, *Data Mining and Knowledge Discovery*, 2007, vol. 14, pp. 245–304.
12. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events, *Journal of Machine Learning Research*, 2009 (10), pp. 1305–1340.
13. Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees, *Artificial Intelligence*, 1998, vol. 101, pp. 285–297.
14. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., van Dongen, B.F., Kindler, Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting, *Software and System Modeling*, 2010 (9), pp. 87–111.
15. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded Petri nets, *IEEE Transactions on Computers*, 2010 (59), pp. 371–384.
16. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming, *Fundamenta Informaticae*, 2009, vol. 94, pp. 387–412.
17. Ferreira, D.R., Gillblad, D.: Discovering process models from unlabelled event logs, in: *Proceedings of the 7th International Conference on Business Process Management* (BPM), 2009, LNCS 5701, pp. 143–158.
18. Sole, M., Carmona, J.: Region-Based Folding in Process Discovery, *IEEE Transactions on Knowledge and Data Engineering*, 2013, vol. 25(1), pp. 192-205.
19. Günther, C.W., Verbeek, E.: XES Standard version 2, 2014, http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf
20. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation, in: Proc. 8th ACM Intern. Conf. Knowl. Discov. Data Mining, ACM (2002), pp. 429–435.