AS-SIM: An Approach to Action-State Process Model Discovery

Please, cite this paper as:

Alessio Bottrighi, Marco Guazzone, Giorgio Leonardi, Stefania Montani, Manuel Striani, Paolo Terenziani.
"AS-SIM: An Approach to Action-State Process Model Discovery."
In: Michelangelo Ceci, Sergio Flesca, Elio Masciari, Giuseppe Manco, Zbigniew W. Raś (eds) Foundations of Intelligent Systems. ISMIS 2022. Lecture Notes in Computer Science, vol 13515, 2022. Springer, Cham. DOI: 10.1007/978-3-031-16564-1_32.

Publisher: https://doi.org/10.1007/978-3-031-16564-1_32

AS-SIM: an approach to Action-State Process Model Discovery

Alessio Bottrighi[1], Marco Guazzone[1], Giorgio Leonardi[1], Stefania Montani[1], Manuel Striani[2] and Paolo Terenziani[1]

[1] DISIT, Università del Piemonte Orientale "A. Avogadro", Alessandria, Italy
[2] Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino, Italy

{alessio.bottrighi, marco.guazzone, giorgio.leonardi, stefania.montani,paolo.terenziani}@uniupo.it
{manuel.striani}@unito.it

Abstract. Process model discovery has gained a lot of attention in recent years, to mine a process model from traces of process executions. In our recent work, we have proposed SIM (Semantic Interactive Miner), an innovative process mining tool able to discover the process model in an incremental way: first, a mining module builds an initial process model, called *log-tree*, from the available traces; then, such a model is refined interactively with domain experts, through *merge* and *abstraction* operations. However, in several contexts, traces are richer: they do not record only actions, but also *states* (i.e., values of parameters possibly affected by the actions). A typical example is the medical domain, where traces contain both actions and measurements of patients' parameters. In this paper, we propose AS-SIM (Action-State SIM), the first approach aiming at discovering a comprehensive model, in which two distinct classes of nodes are considered, to capture both actions and states. We focus on the definition and on the discovery of the initial action-state process model (called *action-state log-tree*), while in our future work we will extend SIM's *merge* and *abstraction* operations accordingly.

Keywords: Process Mining, Process Model Discovery, Mining action+state evolution

1 Introduction

Process Mining (PM) [1] describes a family of a-posteriori analysis techniques able to extract non-trivial information from the *event log*, a repository storing the sequences of actions (*traces* henceforth [1]) that have been executed at a given organization. Within PM, *process model discovery* techniques take as an input the traces and build a process model, focusing on its control flow constructs. The mined process model is an oriented graph (whose semantics can typically be modeled as Petri Nets), where nodes represent trace actions and arcs represent the ordering relation between them. Process model discovery is the most relevant and widely used PM sub-field, and has given birth

to a large family of algorithms. Such approaches have proved to be successful in several applications, such as, e.g., business or production processes. However, in some domains, actions are strictly dependent on the *state* they operate on, and their effect (in terms of the variations they produce on such a state) is not strictly predictable\deterministic. In all such domains, organization logs usually contain also *state* information. As a typical example, in medicine, actions depend on and modify the state of a patient, and patient traces record (possibly timestamped) sequences, where actions are interleaved with state descriptions (consisting of the recording of patient's parameters values), or the information recorded in the hospital information system can be converted into this format (see e.g. [2]). In these cases, mining a model in which only the flow of actions is considered looks reductive, since:

(i) The model would be incomplete, as it would not consider the state in which actions are performed, and their effects on such a state, and
(ii) Such pieces of information are indeed available, and can be mined from the traces and exploited to better characterize the model itself.

In this paper, we propose a new line of process mining, that we call *action-state process model discovery*, in which we extend "traditional" process model discovery to consider also states. Though our proposal is general, and could be carried on as an extension of other miners in the literature, for the sake of concreteness in this paper we describe it as an extension of SIM (Semantic Interactive Miner), a mining algorithm we developed in the past [3].

SIM supports experts in an interactive step-by-step procedure for discovering a process model. Interactive process discovery in SIM starts with an algorithm which mines a process model, called *log-tree*, from the log. The log-tree is already a process model, but possibly an overfitting one, since it perfectly corresponds to the input traces. Experts can move progressively to more generalised graph-based process models, through the application of *merge* and *abstraction* operations. Given a set of occurrences of an activity pattern, *merge* operations modify the current model by "putting them together". On the other hand, *abstraction* operations are based on a-priori knowledge about action decompositions, and support the possibility of abstracting component actions into the macro-actions constituted by them. Each process model can then be evaluated by experts both quantitatively and qualitatively. Through a versioning mechanism, experts can also navigate the history of the versions of the model, generate new versions, or backtrack to a previous one, until they "approve" one of the models.

In this paper we present AS-SIM (Action-State SIM), which extends SIM to deal with state mining as well. In particular, in Section 2 we present our representation formalism. In Section 3, we describe our approach to mine the initial action-state process model. Section 4 is devoted to comparisons and conclusions.

Note that we focus on the definition and on the discovery of the initial action-state process model (called *action-state log-tree*), while in our future work we will extend *merge* and *abstraction* operations accordingly.

2 Action-state log-tree: representation formalism

In a SIM process model, only actions were considered, and each node in the initial log-tree (or in the graph obtained using merge and abstraction operations) could represent either a single action, or a set of actions to be executed in any order. In AS-SIM we consider input traces containing both actions and state information (about a set of parameters). Therefore, the representation formalism needs to be extended, to collect in the ***action-state log-tree*** (and, later, in the action-state graph), also state details.

Notation. We denote by $\mathbb{A}$ the domain $\{a_1, \ldots, a_n\}$ of actions, by $\mathbb{P}$ the domain $\{p_1, \ldots, p_m\}$ of parameters, and by $\mathbb{D}_{P_i}$ the domain $\{v_1, \ldots, v_k\}$ of the values that can be assumed by the parameter p_i (where $\mathbb{D}_{P_i}$ is a discrete and finite domain, as discussed in Section 3.1). Also, given a set T of traces, we denote by $\mathbb{A}_T$ the set of all the actions appearing in the traces in T.

Definition 1 (state). We define ***state*** as a nonempty set of pairs $<p_i,v_j>$ (where $p_i \in \mathbb{P}$ and $v_j \in \mathbb{D}_{P_i}$ and p_i appears at most once) appearing together in the trace (i.e., belonging to the same "observation"). Let $\mathbb{S}$ be the domain of possible states.

Notably, by "state" we refer to a set of variables, measured at a given timepoint, that describe the situation of the entity on which the process at hand operates, e.g., a patient if we work in the medical domain. Indeed, in this case,. according to [5], "states describe clinical situations in terms of a set of state variables with a clinical sense". In real contexts, some state variables may be unavailable at the time of measurement – so the data collection can be incomplete.

In AS-SIM, we distinguish between two types of nodes for action-state log-trees: action-nodes and state-nodes.

Definition 2 (action-node, state-node). ***Action-nodes*** represent a pair $<As,Ts>$, and ***state-nodes*** represent a pair $<Ss,Ts>$, where

- *As* denotes a (possibly unary) set $\{a_1, \ldots, a_k\}$ of actions ($a_i \in \mathbb{A}$, $1 \leq i \leq k$). Actions in the same node are in *any-order* relation.
- *Ss* denotes a (possibly unary) sequence $(s_1, \ldots, s_k)$ of states ($s_i \in \mathbb{S}$, $1 \leq i \leq k$).

Notice that, in such a way, each *path* from the root of the action-state log-tree to a given node *N* denotes a set of possible action-state patterns (called ***support patterns*** of *N* henceforth), obtained by following the order represented by the arcs in the path to visit the action-state log-tree, ordering in every possible way the actions in each action-node, and considering the sequence of states in state-nodes. For instance, the path $\{a,b\} \rightarrow (s_1,s_2) \rightarrow \{c\}$, where $a,b,c \in \mathbb{A}$ and $s_1,s_2 \in \mathbb{S}$ represents the support patterns "a b s_1 s_2 c" and "b a s_1 s_2 c".

- *Ts* represents a set of pointers to all and only those traces (called ***support traces*** henceforth) in the log whose prefixes exactly match the one of the *support patterns* of *As* or *Ss*.

Indeed, action-nodes support a compact representation of sets of actions which can be executed in any order, and state-nodes compact the representation of a sequence of states into a unique node. Two or more actions are in any order only when their order

of execution is not relevant to the process goals. Additionally, we introduce a further type of nodes, macro-state-nodes, as a way to incapsulate alternative state-nodes into a unique node.

Definition 3 (macro-state-node). ***Macro-state-nodes*** represent a set of state-nodes.

3 Mining the action-state log-tree

The mining process takes as an input the event log (i.e., a set of traces), and provides as an output a tree, containing action-nodes and macro-state-nodes (notably, we intend that a macro-state-node may also include a single state-node). In general, input logs can assume different forms. To facilitate the definition of the mining algorithm (section 3.2), we first perform pre-processing operations on the log (section 3.1). Then, we obtain the final action-state log-tree through a post-processing transformation (section 3.3).

3.1 Log pre-processing

In this pre-processing phase, the values of the parameters in the input traces are discretized, according to a set of discretization functions.

Definition 4 (domain-discretization functions). Given the domain $\mathbb{P} = \{p_1, \ldots, p_m\}$ of parameters, defined (in the input traces) over the domains $\{\mathbb{D}_{P1}, \ldots, \mathbb{D}_{PM}\}$, we define domain-discretization functions as a set of functions $f_{ddf} = \{f_1, \ldots, f_m\}$, where, for each $f_i \in f_{ddf}$, f_i is a function that maps values of the domain $\mathbb{D}_{Pi}$ into values of a new, finite domain $\mathbb{D}'_{Pi}$ (i.e., f_i: $\mathbb{D}_{Pi} \rightarrow \mathbb{D}'_{Pi}$).

In the approach described in this paper, we assume that domain-discretization functions are provided as an input to AS-SIM. Notably, however, such functions can be either provided by domain experts, or pre-computed on the basis of the (values of the parameters in the) input traces (e.g., considering the values probability distributions).

To clarify the concepts exposed, we refer to the domain of stroke treatment. In particular, the patients admitted to the stroke unit should be stabilized before starting any treatment. The main parameters to monitor to check the patient's state: temperature, glycemia, diastolic and systolic pressure. Further patient's features are considered to define the treatment strategy, such as age and time since the stroke onset. We assume that the domain-discretization function is defined in terms of discretization levels provided by domain experts. An example is reported in Table 1.

PARAMETER	MEASURE UNIT	DISCRETIZATION LEVELS
Temperature (T)	**° C**	**[0 – 35); [35 – 37.5); [37.5 and beyond)**

Glycaemia (G)	**mg/dL**	**[0 – 50); [50 – 180); [180 and beyond)**
Diastolic pressure (DP)	**mmHg**	**[0 – 60); [60 – 120); [120 and beyond)**
Systolic pressure (SP)	**mmHg**	**[0 – 100); [100 – 185); [185 and beyond)**
Age	**Years**	**[0 – 18); [18 – 45); [45 – 80); [80 and beyond)**
Time since onset (TSO)	**Hours**	**[0 – 4.5); [4.5 and beyond)**

Table 1: discretization levels of patients' parameters

Moreover, in general, each trace in the log consists of a sequence of elements, where each element may be either an action or a state, with no constraint. In particular, also sequences of states may appear in the traces. To facilitate the construction of state-nodes (see Section 2), in the pre-processing phase, we merge sequences of states into a unique element, modeling the sequences of values assumed by the parameters, at each state.
For example, during the patient stabilization phase the monitoring data are collected periodically to check the patient's state, in order to take actions accordingly. This can generate sequences of states in the traces; Figure 1 shows an excerpt of a trace containing a state sequence.

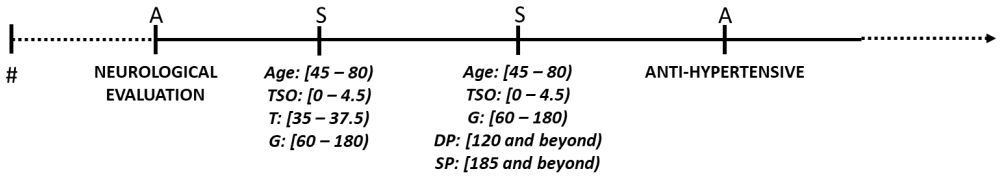

Figure 2: excerpt of trace containing sequence of states

In Figure 1, each vertical line represents an action (identified by the character "A" above the line) or a state (character "S"). After the pre-processing phase, the same trace will be arranged as shown in Figure 2.

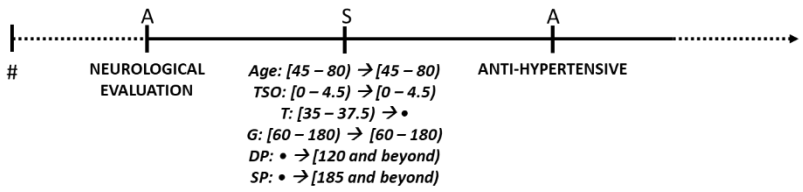

Figure 2: trace in Figure 1 after pre-processing

In Figure 2, the two consecutive states in Figure 1 are merged into a unique element. In this merged state, each parameter is represented as a sequence, where not measured data are indicated with the value "•".

3.2 Mining algorithm

In this section, we present the algorithm to build the action-state log-tree as an extension of the one used in SIM [3]. The pseudocode is shown in Algorithm 1. The function Build-Tree in Algorithm 1 takes as input a variable index, representing a given position in the traces, a node ⟨P,T⟩ (either an action-node or a state-node), and two user-defined thresholds α and β (for details, see below). Initially, it is called on the first position in the traces (we assume that all the traces start with a dummy action #) and on the root of the action-state log-tree (which is a dummy node, corresponding to the # action; therefore, initially, index=0, P={#} and T is the set of all the traces).

Algorithm 1: *Build-Tree* pseudocode.

```
1. Build-Tree(index, ⟨P,T⟩, α, β)
2. ⟨nextP_S, nextP_A⟩ ← getNext(index+1, T, α)
3. if (nextP_S ∪ nextP_A) not empty then
4.   nextNodes ← XORvsANY(⟨nextP_S, nextP_A⟩, T, β)
5.   foreach node <P',T'> ∈ nextNodes do
6.     AppendSon(⟨P',T'⟩, ⟨P,T⟩)
7.     Build-Tree(index+|P'|, ⟨P',T'⟩, α, β)
8.   end
9. end
```

The function *getNext* (see line 2) inspects the traces in T to find all possible next elements (either actions or states). At this stage, "rare" patterns can be ruled out. Specifically, if $P=\{X\}$, and Y is a possible next element, Y will be provided in output by *getNext* only if the edge frequency E_F of the sequence $X>Y$ is above a user-defined threshold α, where:

$$E_F(X > Y) = \frac{|X > Y|}{|T|}$$

being $|X>Y|$ the number of traces in T in which X is immediately followed by Y (i.e., the cardinality of the support traces of Y), and being $|T|$ the cardinality of the support

traces of X. Setting $\alpha > 0$ allows to rule out noisy patterns. Note that, if rare/noisy patterns are ruled out, the resulting action-state log-tree is not guaranteed to still have precision=1 and replay-fitness=1 (see [4]); however, in this paper, we will set α=0, and thus consider all the traces in the initial model construction.

On the remaining next states $nextP_S$ and next actions $nextP_A$, the function *XORvsANY* identifies possible sets of actions in any-order (see line 4). Notably, (i) we assume that states cannot be in any-order (so they are directly managed as XOR sons of the current node), and (ii) the same action may appear more than once as a son of the current node (e.g., as a "unitary" action-node and\or in one or more "any-order" action-nodes).

To identify any-order sets of actions appearing in $nextP_A$, support traces in T are inspected at positions *index+2* (binary any-order), *index+3* (ternary any-order) and so on, until no "wider" any-order can be determined. For the sake of simplicity, let us consider the case of binary any-orders: *XORvsANY* calculates the dependency frequency $A \rightarrow B$ between every action pair $\langle A,B \rangle$ in $nextP_A \times nextP_A$ by considering sequences of two actions A (at position *index+1*) and B (at position *index+2*) following P in the traces T as follows:

$$A \rightarrow B = \frac{1}{2}\left(\frac{|A > B|}{\sum_{C \in A_T} |A > C|} + \frac{|A > B|}{\sum_{D \in A_T} |D > B|}\right)$$

where, always considering the traces in T, $|A > B|$ is the number of traces in which A is immediately followed by B, $|A > C|$ is the number of traces in which A is immediately followed by some action $C \in \mathbb{A}_T$, and $|D > B|$ is the number of traces in which B is immediately preceded by some action $D \in \mathbb{A}_T$. If both the dependency frequencies of $A \rightarrow B$ and $B \rightarrow A$ are above the given (user-defined) threshold β, this means that A and B occur frequently in any-order in the same traces. Thus, *XORvsANY* identifies an any-order relation between A and B, and creates a new node $A\&B$ with the associated support traces.

The output *nextNodes* of the function *XORvsANY* is a set of nodes $\langle P', T' \rangle$. Each node is appended to the action-state log-tree (function *AppendSon*; see line 6), and *BuildTree* is recursively applied to each node (with the first parameter *index* properly set according to the cardinality of *P'*; see line 7).

Finally, a-posteriori, we create a dummy node \$, and connect all the leaves to it.

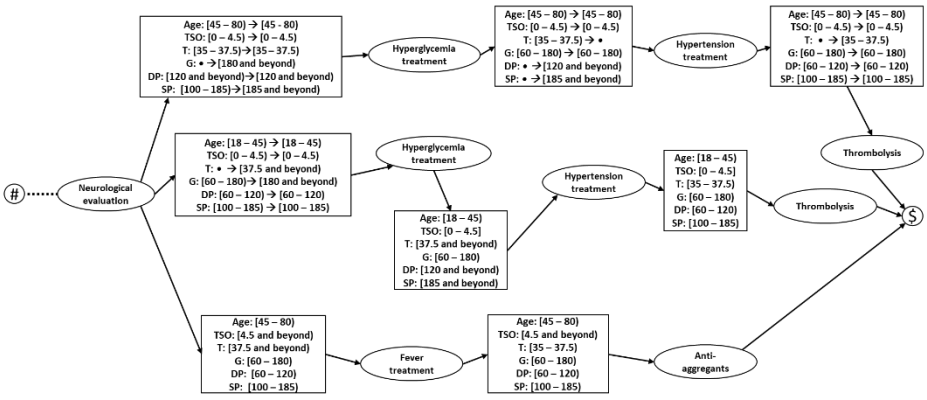

Figure 3: action-state log-tree for patients' stabilization in stroke disease

Figure 3 shows the action-state log tree, based on traces from the stroke domain focusing on the patients' stabilization phase described in Section 3.1. Here, the states are represented by rectangles, while the actions by ellipses. In the process in Figure 3, after the neurological evaluation, different stabilization strategies are performed (including Hyperglycemia, Fever or Hypertension treatments) on the basis of the patients' states, before adopting the most appropriate treatment (Thrombolysis or Anti-aggregants).

3.3 Model post-processing: generating macro-state-nodes

The action-state log-tree, as modeled so far, represents in a compact way the input traces. However, an additional simplification can be provided. Given the fact that, in many real situations, the number of state parameters may be high, as well as the number of values they may assume (even after discretization), states deriving from different traces are rarely identical, so that the action-state log-tree "spans" in many different branches whenever states appear. We propose, as a first "compacting" step, to automatically merge all the state-nodes in the action-state log-tree which are preceded by the same action-node and are followed by the same action-nodes, into a unique macro-state-node, representing their union.

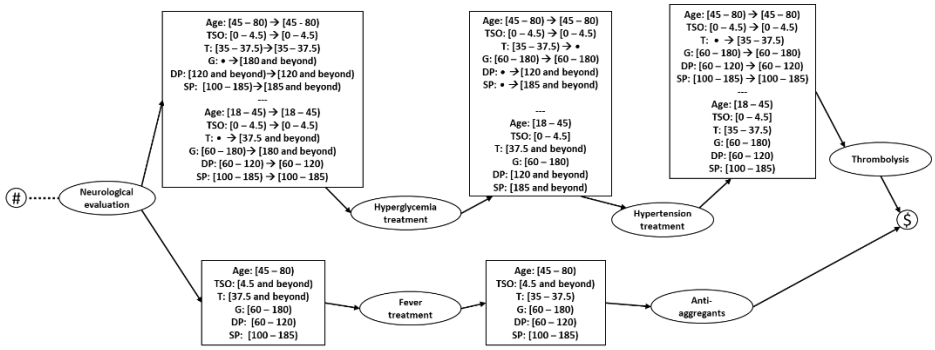

Figure 4: action-state log-tree in Figure 3 after post-processing

Figure 4 shows the action-state log tree in Figure 3, after the post-processing step. In particular, the two upper branches are composed by the same sequence of actions, therefore the post-processing algorithm merges the two state-nodes between "Neurological evaluation" and "Hyperglycemia treatment"; the state-nodes between "Hyperglycemia treatment" and "Hypertension treatment" and the state-nodes between "Hypertension treatment" and "Thrombolysis".

A possible interpretation of this generalization is that the same treatment strategy can be applied for patients having slightly different conditions.

Notably, besides making models more compact and "readable", the introduction of macro-state-nodes facilitates further processing stages (operating on states), as discussed in future work (see section 4).

3.4 Properties of the action-state log-tree model

The action-state log-tree is a tree, and has precision = 1 (i.e., each path in the action-state log-tree corresponds to at least one trace in the log), and replay-fitness = 1 (i.e., each trace in the log can be replayed in the action-state log-tree with no errors) (see [4]).

4 Comparisons, future work and conclusions

In this paper, we have proposed the first process model discovery approach mining and providing an explicit representation of the flow of both actions and states from input traces. Indeed, some approaches have started to face such an important issue. In the area of medical data mining, Kamisalic et al. [5] have considered the case of traces representing sequences of patient visits. Each visit assesses the state of a patient, and is followed by a prescription (indicating the therapies to be followed until the next visit). From this input, they mine a graph, where nodes represent patient' states, and arcs are labeled by the therapies leading from the input to the output states. With respect to our approach, therefore, they focus on states and on their transitions, not on the overall action-state process model.

In the PM literature, it is worth mentioning the work by De Leoni et al (see, e.g., [6, 7]), that takes into account not only the (actions) control flow perspective, but also the data flow one. Specifically, they analyze the data flow to find rules explaining why individual cases take a particular path, i.e., they explain the behavior of process instances with respect to decision points in the model on the basis of observed data. To this end, they resort to conformance checking (another PM sub-field) techniques to align an event log containing data information and a process model with decision points. These alignments are then used to generate a classification problem, afforded by decision trees. The output is an extended Petri Net, where guards are introduced to describe the effect of data on transitions, i.e., what actions have to be executed on the basis of data values.

Notably, however, none of the above approaches has explicitly addressed the general problem of considering input traces in which action and state information are provided, to discover a process model explicitly distinguishing between action and state nodes. Such a generalized model is, in our opinion, extremely important in all domains where the effects of actions on states have to be analysed. We therefore believe that our approach may become the starting point of a new stream of research in the area of process model discovery.

In the future, we plan to extend the work along several lines.

First of all, we will conduct an extensive experimental evaluation. Though the proposed methodology is general, we will consider the medical context, where we have been operating since a long time. In particular, we will resort to the public available patient data in the Mimic database [8].

Other major evolutions regard the discovery process. The action-state log-tree is already a process model, but possibly an overfitting one, since it perfectly corresponds to the input traces (see Section 3.4). Experts may want to move progressively to more generalised graph-based process models. In SIM, considering only actions, such

generalizations could be obtained through the application of a wide class of *merge* and *abstraction* operations. In the future, we aim at extending AS-SIM along these directions, and more specifically we will work on the:

- Definition of *intra-state generalization* operations. Such operations will apply to macro-state-nodes, to discover hidden pieces of information (e.g., to abstract parameter trends from sequences of states) and to use them in order to simplify the state representation, making it more "compact";
- Definition of *merge* and *abstraction* operations, generalizing and complementing the current operations in SIM in order to consider also state-nodes.

References

1. Aalst, W.M.P. van der: Process Mining - Data Science in Action, Second Edition. Springer (2016). https://doi.org/10.1007/978-3-662-49851-4.
2. Wang, S., McDermott, M.B.A., Chauhan, G., Ghassemi, M., Hughes, M.C., Naumann, T.: MIMIC-Extract: a data extraction, preprocessing, and representation pipeline for MIMIC-III. In: Ghassemi, M. (ed.) ACM CHIL '20: ACM Conference on Health, Inference, and Learning, Toronto, Ontario, Canada, April 2-4, 2020 [delayed]. pp. 222–235. ACM (2020). https://doi.org/10.1145/3368555.3384469.
3. Bottrighi, A., Canensi, L., Leonardi, G., Montani, S., Terenziani, P.: Interactive mining and retrieval from process traces. Expert Systems with Applications. 110, 62–79 (2018). https://doi.org/10.1016/j.eswa.2018.05.041.
4. Buijs, J., Dongen, B. van, Aalst, W. van der: On the role of fitness, precision, generalization and simplicity in process discovery. In: On the Move to Meaningful Internet Systems: OTM 2012. pp. 305–322. Springer (2012).
5. Kamisalic, A., Riano, D., Welzer, T.: Formalization and acquisition of temporal knowledge for decision support in medical processes. Comput Methods Programs Biomed. 158, 207–228 (2018). https://doi.org/10.1016/j.cmpb.2018.02.012.
6. Leoni, M. de, Aalst, W.M.P. van der: Data-aware process mining: discovering decisions in processes using alignments. In: Shin, S.Y. and Maldonado, J.C. (eds.) Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013. pp. 1454–1461. ACM (2013). https://doi.org/10.1145/2480362.2480633.
7. Leoni, M. de, Felli, P., Montali, M.: A Holistic Approach for Soundness Verification of Decision-Aware Process Models. In: Trujillo, J., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., and Lee, M.-L. (eds.) Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings. pp. 219–235. Springer (2018). https://doi.org/10.1007/978-3-030-00847-5_17.
8. Johnson, A.E.W., Pollard, T.J., Shen, L., Lehman, L.H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., Mark, R.G.: MIMIC-III, a freely accessible critical care database. Scientific Data. 3, 160035 (2016). https://doi.org/10.1038/sdata.2016.35.