data

MDPI

Article

Towards Action-State Process Model Discovery

Alessio Bottrighi [1,2,†], Marco Guazzone [1,3,†], Giorgio Leonardi [1,2,†], Stefania Montani [1,2,†], Manuel Striani [1,2,*,†] and Paolo Terenziani [1,2,†]

[1] Department of Science, Technology and Innovation, Università del Piemonte Orientale, Viale Teresa Michel 11, 15121 Alessandria, Italy; alessio.bottrighi@uniupo.it (A.B.); marco.guazzone@uniupo.it (M.G.); giorgio.leonardi@uniupo.it (G.L.); stefania.montani@uniupo.it (S.M.); paolo.terenziani@uniupo.it (P.T.)
[2] Laboratorio Integrato di Intelligenza Artificiale e Informatica Medica DAIRI, Azienda Ospedaliera SS. Antonio e Biagio e Cesare Arrigo, Alessandria e DISIT—Università del Piemonte Orientale, 15121 Alessandria, Italy
[3] Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), 43124 Parma, Italy
* Correspondence: manuel.striani@uniupo.it; Tel.: +39-0131360179
† These authors contributed equally to this work.

Abstract: Process model discovery covers the different methodologies used to mine a process model from traces of process executions, and it has an important role in artificial intelligence research. Current approaches in this area, with a few exceptions, focus on determining a model of the flow of actions only. However, in several contexts, (i) restricting the attention to actions is quite limiting, since the effects of such actions also have to be analyzed, and (ii) traces provide additional pieces of information in the form of states (i.e., values of parameters possibly affected by the actions); for instance, in several medical domains, the traces include both actions and measurements of patient parameters. In this paper, we propose AS-SIM (Action-State SIM), the first approach able to mine a process model that comprehends two distinct classes of nodes, to capture both actions and states.

Keywords: process mining; process model discovery; mining action-state evolution

check for updates

Citation: Bottrighi, A.; Guazzone, M.; Leonardi, G.; Montani, S.; Striani, M.; Terenziani, P. Towards Action-State Process Model Discovery. *Data* **2023**, *8*, 130. https://doi.org/10.3390/data8080130

Academic Editors: Edson Talamini, Letícia De Oliveira and Filipe Portela

Received: 27 June 2023
Revised: 1 August 2023
Accepted: 3 August 2023
Published: 9 August 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Process mining (PM) [1] consists of different methodologies of a posteriori analysis, able to take as input a repository (termed *event log)* and storing the sequences of actions (*traces* henceforth [1]) that have been executed at a given organization, to extract non-trivial information. *Process model discovery* is one of the main sub-areas of PM and aims at mining a process model from an event log. A process model is a directed graph, where nodes represent actions in the traces and arcs represent the ordering relation between them. Many different process discovery methodologies have been proposed in the literature, obtaining successful results in many application domains, such as business or production processes.

However, while in many contexts a "traditional" process model representing the flow of actions suffices for including all the key elements for further analysis, in other cases, the actions cover only one part of the useful information, since their enactment depends on the current state of affairs and their effect (in terms of the variations they produce in such a state) is not strictly predictable/deterministic. It is therefore essential to also include a description of the evolution of the state in the model. In all such applications, event logs usually also contain state information. For instance, in a medical context, actions are poorly meaningful if they are not related to the current state of the patient, and considering their effects (in terms of modifications to the previous state of the patient) is a crucial issue. In fact, patient traces log sequences of (timestamped) actions and states (where a state is the recording of a patient's parameters), or the information stored in the hospital information system can be converted into this format (see e.g., Ref. [2]).

In these domains, mining a model considering only the flow of actions is limiting, since

i. the model would be incomplete, as it would neglect the state where actions are performed, and actions' effects on such a state, and
ii. state information are indeed available in the input traces and can therefore be exploited.

In this paper, we aim to introduce a new line of research in process model discovery, which we term *action–state process model discovery*, to extend "traditional" process model discovery techniques to also cope with state-related information.

The key contributions of our proposal are general and could be applied to extend different process miners in the literature. However, for the sake of concreteness, in this paper, we specifically work to extend SIM (semantic interactive miner), a mining algorithm we have recently developed [3,4]. With respect to other miners in the literature, SIM is characterized by the fact that it supports highly interactive (with analysts/domain experts) step-by-step sessions of work to discover the process model, starting from a basic automatically mined model with maximal precision [5] (i.e., fully adherent to the event log content), and then progressively generalizing/simplifying it through the application of *merge* and *abstraction* operations (see Section 2).

As usually happens in this area of research, the objective of our approach is the mining a specific process model from a given event log; however, the approach is general enough to be applied to different domains, and its usage can cover a wide variety of situations, each one described by its own event log; and the corresponding process model will be mined. This approach would be particularly useful in those situations where the state information is relevant and impacts on the action flow.

In the following, we thus present AS-SIM (action-state SIM), which extends SIM to also deal with state mining. In particular, in Section 2, we briefly overview SIM. In Section 3, we present our representation formalism: a bipartite graph consisting of both action nodes and state nodes. In Section 4, we outline the general behavior of AS-SIM and its basic functionalities. In Section 5, we describe our approach to the pre-processing of the log and mining the action-state log. In Section 6, we propose a methodology to achieve a compact representation of state nodes (possibly interacting with analysts). In Section 7, we present an experimental evaluation to assess system performance. In Section 8, we discuss related work. The extensions of SIM's *merge* and *abstraction* facilities to operate with the new bipartite process model are left as future work, as discussed in Section 9.

2. Overview of SIM

In this section, we provide an overview of the main features of SIM. For more details, please refer to our previous works [3,4].

In the area of process discovery, many systems are designed as "black boxes", where an event log is inputted and a model is outputted based on specific parameters. If the analysts are unsatisfied with the outputted model, they can only re-run the system using different parameters, resulting in a "blind search", as they are unaware of the effect of parameter values on the output model. Instead, SIM seeks to provide a solution by enabling analysts to leverage their knowledge in model discovery and also use pre-encoded domain knowledge (where possible).

SIM provides the possibility of refining (ISA relations) and composing (Part-Of relations) process actions through (taxonomic) a priori domain knowledge. This begins with automatically mining an initial process model from the event log with the precision and replay-fitness [5] set to 1. Although generalizations may be required due to incomplete logs, SIM aims to ensure that these generalizations are driven by analysts via an interactive and incremental process. This is achieved by applying a set of operations to generalize the initial model in different steps.

In synthesis, four main aspects characterize SIM's definition and behavior:

1. *Initial model*: SIM starts by mining an initial model from the log with precision = 1 and replay-fitness = 1. This model exactly covers all the behavior described by the input traces in the logs. However, since the logs are incomplete, some forms of generalization are needed. Thus, generalizations are directly driven by the analyst through an interactive and incremental process, to avoid losing precision and/or replay-fitness;
2. *Generalizations*: In SIM, generalizations can be obtained by applying two types of operations: merge operations and abstraction operations. Merge operations merge the instances of some paths occurring in the model, and SIM provides facilities to specify paths, search the occurrences in the model, and merge them. Abstraction operations move from a process description, where actions are reported at the ground level to more user-interpretable/compact descriptions, in which sets of actions are abstracted into "macro-actions" subsuming them (ISA relations) or constituted by them (Part-Of relations);
3. *Incremental and selective application of generalization operators*: SIM provides analysts with the possibility of selectively applying generalization (e.g., the analyst can decide which occurrences of the path to merge or which instances of the actions composing a macro-action to abstract), operating step-by-step. In this way, the final model can be built by applying subsequent merge and abstraction steps, leading to a progressively more generalized version.
4. *Model evaluation and versioning*: SIM enables analysts to perform a quantitative evaluation of the model, considering parameters such as replay-fitness, precision, and generalization. This also allows analysts to backtrack to previous versions of the model (versioning).

3. Action-State Log-Tree: Representation Formalism

The starting point in the definition of the AS-SIM formalism is the definition of the domains of actions and states.

As in SIM, the domain $\mathcal{A}$ of actions is simply the set of all the actions appearing in the input traces (possibly, a preprocessing phase can be adopted to filter out actions appearing below a given threshold frequency). On the other hand, in AS-SIM, states are described in terms of the values assumed by a set of parameters (i.e., features) measured at a given point of time as reported in the traces, and describing (possibly in a partial way) the context where actions are executed.

For the sake of generality, in AS-SIM, we distinguish parameters along two different dimensions:

- along the *temporal dimension*, a parameter may be *constant* (in cases where the value of the parameter cannot vary in time) or *changing* (in this case, the parameter assumes the form of a time series);
- along the *value-type dimension*, a parameter may be *numeric* (in such a case, the measuring unit adopted for the parameter must be part of the domain description), *symbolic* (i.e., assume a finite set of non-numeric values that cannot be ordered along a scale; e.g., true/false) or *categorical* (i.e., assume non-numeric values that are ordered along a scale; e.g., low < medium < high).

Independently of the value-types, every single value for a parameter represents a measurement/observation on the parameter and is modeled using a pair $\langle value, timestamp \rangle$.

Notation. We denote by $\mathcal{A}$ the domain $\{a_1, \ldots, a_n\}$ of actions, by $\mathcal{P}$ the domain $\{p_1, \ldots, p_m\}$ of parameters, and by PD_i the domain $\{v_1, \ldots, v_{k_i}\}$ of the values that can be assumed by the parameter p_i. In addition, we denote by $\mathcal{T}$ the domain of time (our approach is independent of the chosen time granularity).

Each state is characterized by the values assumed by a nonempty subset of the parameters in $\mathcal{P}$. For instance, in the context of medical traces, "states describe clinical situations

in terms of a set of state variables with a clinical sense" [4]. In real contexts, some state variables may be unavailable at the time of measurement, so that only a subset of parameters in $\mathcal{P}$ is considered.

Definition 1 (State). *A **state** is a nonempty set of pairs $\langle p_i, \langle\langle v_1, t_1\rangle, \ldots, \langle v_j, t_j\rangle\rangle\rangle$ in which $p_i \in \mathcal{P}$ represents a parameter and $\langle\langle v_1, t_1\rangle, \ldots, \langle v_j, t_j\rangle\rangle$ with $j > 0$ is a timestamped series of observations of values $(v_1, \ldots, v_j) \in PD_i$ for p_i and with $t_1 < \cdots < t_j$, where $t_1, \ldots, t_j \in \mathcal{T}$. Domain of possible states is denoted by $\mathcal{S}$.*

For the sake of clarity, we assume the conventions that

- the (unique) value of constant parameters is collected in the initial state (which may possibly also contain values for some of the changing parameters);
- each state (except the initial one) follows a source action (or a set of actions) and precedes a destination action (or set of actions), and contains, for each observed parameter, all the $\langle value, timestamp\rangle$ observations collected for the parameter in the slice of time between the execution of the source action and the destination action.

AS-SIM adopts two types of node: action nodes and state nodes. Action nodes support a compact representation of sets of actions, which can be executed in any order and are defined as follows:

Definition 2 (Action node). *An **action node** represents a pair $\langle A_s, T_s\rangle$ where*

- *A_s denotes a nonempty set $\{a_1, \ldots, a_k\}$ of actions ($a_i \in \mathcal{A}, 1 \leq i \leq k$). Actions in the same node are in any-order relation;*
- *T_s is the set of support traces in the log (*see Definition 6 below*).*

The domain of action nodes is denoted by AN.

State nodes provide a compact representation of a timestamped sequence of parameter observations (i.e., all the observations between two action nodes) into a unique node.

Definition 3 (State node). *A **state node** represents a pair $\langle ST, T_s\rangle$, where*

- *$ST \in \mathcal{S}$ denotes a state (*see Definition 1*);*
- *T_s is the set of support traces in the log (*see Definition 6 below*).*

The domain of state nodes is denoted by SN.

In AS-SIM process models, oriented arcs connect a source node and a destination node to model a temporal sequence relation (i.e., ordering) between them. However, since for the sake of compactness we model a series of observations of parameter values (with no action interleaved between them) into a unique state node, in the AS-SIM formalism we only admit arcs connecting

- two actions nodes $N1$ and $N2$ (to represent the fact that the action(s) in $N1$ precede(s) those in $N2$);
- an action node $N1$ to a state node $N2$ (to represent the fact that the action(s) in $N1$ precede(s) the state described in $N2$);
- a state node $N1$ to an action node $N2$ (to represent the fact that the state in $N1$ precedes the action(s) in $N2$).

On the other hand, arcs starting from a state node and ending in a state node are not allowed. Thus, in AS-SIM, a process model graph is a directed graph, in which two classes of node (action nodes and state nodes) are connected by oriented arcs, as discussed above. In the graph, an initial node can be distinguished.

Definition 4 (AS-SIM process-model graph). *In AS-SIM, a **process-model graph** is a structure $G = \langle Nodes, Arcs\rangle$ where $Nodes \subseteq (AN \cup SN)$ is the set of nodes and $Arcs = \{\langle x, y\rangle | x, y \in$*

Nodes $\wedge$ $((x \in AN \wedge y \in AN) \vee (x \in AN \wedge y \in SN) \vee (x \in SN \wedge y \in AN))\}$ *is the set of arcs (restricted as discussed above).*

In particular, the initial process model (which is automatically mined by AS-SIM, as discussed in [4]) has the form of a rooted tree (i.e., a tree with a unique root), which we call ***action-state log-tree***. As in SIM, in AS-SIM each node in the ***log-tree*** also contains pointers to all the input traces "corresponding" to it (called support traces), as defined below.

Definition 5 (Support patterns)**.** *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 ordering 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 parameter values in state nodes.*

For instance, the path

$$N1 : \{a, b\} \rightarrow N2 : \{\langle p_i, \langle\langle v_1, t_1\rangle, \ldots, \langle v_k, t_k\rangle\rangle\rangle\langle p_j, \langle\langle v_1, t_1\rangle, \ldots, \langle v_h, t_h\rangle\rangle\rangle\} \rightarrow N3 : \{c\}$$

where $N1$ and $N3$ are action nodes (with $a, b, c \in \mathcal{A}$) and $N2$ is a state node, represents the support patterns $a\ b\ \{\langle p_i, \langle\langle v_1, t_1\rangle, \ldots, \langle v_k, t_k\rangle\rangle\rangle\langle p_j\langle\langle v_1, t_1\rangle, \ldots, \langle v_h, t_h\rangle\rangle\rangle\}\ c$ and $b\ a\ \{\langle p_i, \langle\langle v_1, t_1\rangle, \ldots, \langle v_k, t_k\rangle\rangle\rangle\langle p_j, \langle\langle v_1, t_1\rangle, \ldots, \langle v_h, t_h\rangle\rangle\rangle\}\ c$.

Definition 6 (support traces)**.** *Given a node N in an action-state log-tree, the* ***support traces*** *of N are all and only those input traces in the log whose prefixes exactly match one of the support patterns of the node N.*

Besides the *process-model graph*, a process model in AS-SIM should also account for the description(s) of the properties of the parameters used to describe states. Thus, a set of **parameter tables** is also introduced. Notably, as we will detail in the following, certain properties of parameter representation can be changed (interactively with the analysts) during the mining process. As a consequence, multiple parameter tables are defined in our process of model representation.

Each **parameter table** is linked to one or more state nodes and details the properties of parameters. For each parameter, it contains the following properties:

- Temporal Type: constant, changing;
- Value Type: numeric, symbolic, categorical;
- Parameter Operator (having as value NULL, if no discretization/time series reduction has been performed, or the list of the applied operators—see Section 6.1—otherwise).

Additional properties depend on the value-type of the parameters:

- numeric parameters have properties: *range*, *measure unit*;
- symbolic parameters have the *valueset* property, specifying the set of the possible values they may assume;
- categorical parameters have properties *range*, *measure unit*, *discretization function* (see Section 5).

4. An Overview of AS-SIM

Overall, the architecture of AS-SIM is as depicted in Figure 1 and relies on the following workflow (where each number denotes a phase of the workflow):

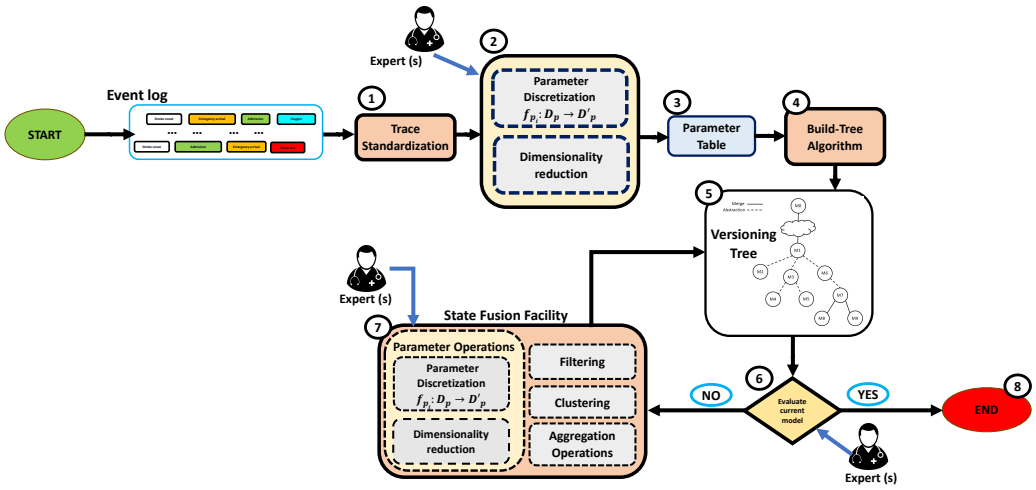


Figure 1. The tool workflow (circled numbers represent the workflow phases presented in Section 4).

1. the event log is a file that stores sequences of activities (called "process traces") that have been executed at a given organization (e.g., a hospital), typically together with some key parameters, such as execution times, resources, and originators (who performed those particular actions); the event log is inputted to **Trace Standardization** (see Section 5);
2. then, with the help of the domain expert(s), in the second step of log pre-processing, the standardized process traces can be transformed (optionally) by applying **parameter discretization** (for constant parameters) and **dimensionality reduction** (for changing parameters), see Section 5;
3. the **parameter table**, which contains the details of parameter properties, as described in Section 3, is built as output of the pre-processing phase and appropriately updated on the basis of the pre-processing operations illustrated at phases 1 and 2. In this phase, the parameter table includes parameters and characteristics of the process model at the global level; it will then be linked to the root of the **versioning tree** in phase 4 (see below);
4. the **Build-Tree Algorithm** (see [4]) constructs the initial action-state process model, starting from the pre-processed log, and the versioning tree root is created;
5. the workflow now enters a loop: at every iteration, a new process model, i.e., a new node in the versioning tree, is generated, along the lines described in the following steps: The versioning tree captures the evolution of the process model from the initial action-state log-tree; backtracking can possibly be applied, and a different model generalization strategy can be tested;
6. the expert, on the basis of domain knowledge, evaluates the goodness of the current model. If the model does not need any further operations, then the workflow ends (go to step 8), otherwise it proceeds to step 7, remaining in the model revision loop;
7. on the basis of domain knowledge and by using the set of facilities described in Section 6, the expert can collapse the set of states occurring between the same two actions A and B, thus improving the readability and simplicity of the model. In particular, s/he can take advantage of the same **parameter operations** (divided into *parameter discretization* and *dimensionality reduction*) adopted in phase 2; additionally, s/he can rely on *filtering* and *clustering operations*. Finally, **aggregation operations** perform the actual fusion of the state nodes and allow the user to aggregate the different values of the parameters collected in different state nodes into single summary representations. These (optional) facilities can be differently combined, acting as the building blocks of the overall model generalization strategy; each node of the versioning tree is associated with a process model, and with a parameter table for every state node in the model, which is properly updated in the state fusion step;

8. the workflow ends when the domain expert evaluates the process model as satisfactory, without the necessity of applying additional generalization facilities.

In the following, a more detailed description will be provided of the workflow phases presented above.

5. Pre-Processing of the Log and Mining the Action-State Log-Tree

The initial process mining step in AS-SIM takes as input an event log (i.e., a set of traces) and provides as an output a tree containing action nodes and state nodes (action-state log-tree). The interested reader will find details of the mining algorithm in [4]. In general, input logs can assume quite different formats; therefore, we first pre-process the log to "standardize" and possibly summarize its information (Section 5).

Log Pre-Processing

Given the variety of formats for parameters in organization event logs, we introduce a compulsory pre-processing phase to "standardize" them.

Then, optionally, the traces in the log can be transformed by applying

- discretization of constant parameters;
- dimensionality reduction in changing parameters (i.e., time series).

Discretization is achieved through the definition and application of a set of discretization functions. Specifically, for each numeric parameter $p_i \in \mathcal{P}$ (with domain D_p) we define a discrete domain of values $D_p^{'}$ and a total discretization function $f_{p_i} : D_p \rightarrow D_p^{'}$ that maps any value in D_p onto a corresponding value in $D_p^{'}$.

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

When working on time series, on the other hand, keeping all the sampled parameter values may be useless: more thoroughly summarized information would save storage space, while at the same time clarifying the series behavior. To this end, SIM incorporates a set of *dimensionality reduction* techniques. Currently, we provide a set of mathematical transforms that move from the time domain to the time domainbut reduce dimensionality, namely the discrete wavelet transform and piecewise aggregate approximation (PAA) [6].

Wavelets are basis functions used to represent other functions, which are local both in time and frequency. The wavelet transform can be repeatedly applied to the data; each application brings out a higher resolution of the data, while at the same time, smoothing the remaining data.

Through PAA, on the other hand, the original time series is divided into intervals of equal length and is transformed in a piecewise constant function, where all the time points within the same interval assume as a value the average of the original time series values in the interval itself.

In addition to such transforms, we also provide a more qualitative dimensionality reduction technique, namely temporal abstractions (TA) [7].

TA is an Artificial Intelligence technique that derives high-level concepts from time-stamped data. It allows one to move from a point-based representation of time series data to an interval-based one, where the input points are the sampled measurements and the output intervals aggregate adjacent points sharing a common behavior, persistently over time. Such a behavior is identified by an appropriate qualitative symbol. Basic temporal abstractions can be further subdivided into state TAs and trend TAs. State TAs are exploited to extract intervals associated with qualitative levels, such as low and high parameter values; trend TAs are used to detect intervals of increasing or decreasing behavior. Through TA, huge amounts of temporal information can be effectively mapped to a compact symbolic

representation, which not only summarizes the original longitudinal data but also maps them to a qualitative domain with clear semantics.

Such operations can also be applied on parameters in the phase of the fusion of state nodes (see Section 6).

Example 1. *To clarify the above concepts, we consider the domain of stroke treatment, focusing in particular on patients eligible for thrombolysis treatment (TPA). Although this treatment is particularly effective in breaking down the blood clots that cause ischemic attack, it can be dangerous for patients, who need continuous monitoring of their vital signs before and after the administration of the TPA drug. In particular, among the parameters to be taken into account to define patient states, the most important are age and time since stroke onset (TSO). For these parameters, the domain-discretization function is defined in terms of discretization levels provided by domain experts, as reported in Table 1. Among the monitoring parameters, particular attention is paid to assessing the patients' cardiac and respiratory status; therefore, we focus on arterial pressure (AP), cardiac frequency (CF), and respiratory frequency (RF), which are collected in the form of time series.*

Table 1. Discretization levels of selected patients' parameters.

Parameter	Measure Unit	Discretization Levels
Age	Years	[0–18); [18–45); [45–80); [80 and beyond)
Time since onset (TSO)	Hours	[0–4.5); [4.5 and beyond)

6. Fusion over State Nodes

An action-state log-tree can often be quite wide. In particular, in many cases, a wide set of state nodes $\{S_1, \ldots, S_k\}$ may lay between an "origin" action node A and the "destination" action node B: in the log-tree, different instances of B, one per branch, are represented. This makes the log-tree redundant and difficult to read. Therefore, in this case, a simple post-processing step is adopted, to merge all such instances of B into a single action node, technically transforming the log-tree into a graph (action-state base-graph henceforth).

Example 2. *Figure 2 shows an action-state base-graph, based on traces from the stroke domain, focusing on the TPA administration procedure, as described in Section 5. This model was obtained after applying the trace standardization procedure to the raw event log (step 1 of the workflow in Figure 1), the parameter discretization step (step 2 of the workflow), and launching the Build-Tree algorithm on the obtained pre-processed log, using the relative parameter table (steps 3 and 4 of the workflow). In the obtained model, shown in Figure 2 and corresponding to node M0 of the versioning tree in Figure 3a, the states are represented by rectangles, while the actions are ellipses. In this process, after admission to the stroke unit (SU), the patients are evaluated by experts and the TPA drug is administered. At the conclusion of the TPA infusion, the patients are monitored and sent to the neurological ward or further examined through the installation of a continuous ECG monitor before being discharged to the sub-intensive care ward. In the neurology ward, the model in Figure 2 suggests that patients can undergo an additional treatment (thrombectomy) or be discharged to the sub-intensive care ward. For brevity and readability, we do not specify the contents of the state nodes in Figure 2.*

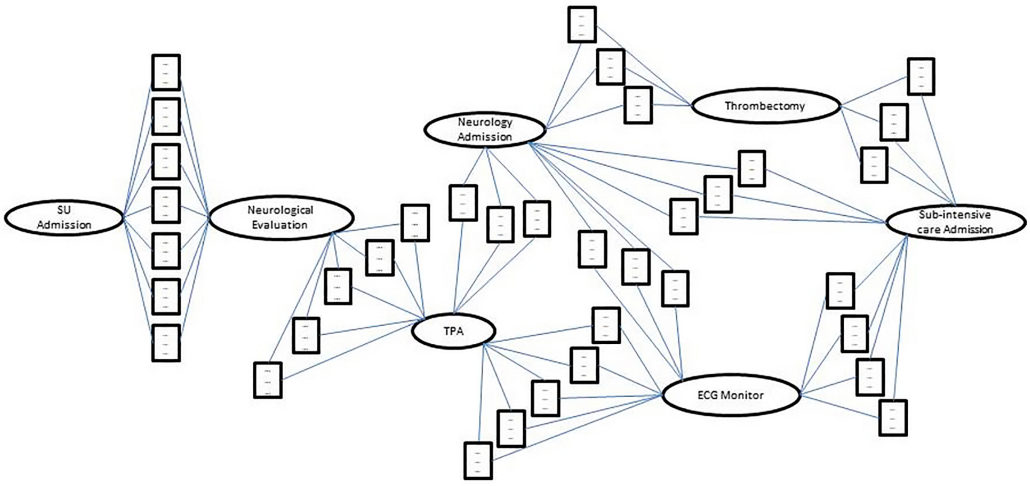


Figure 2. Action-state base-graph for thrombolysis treatment of stroke disease.

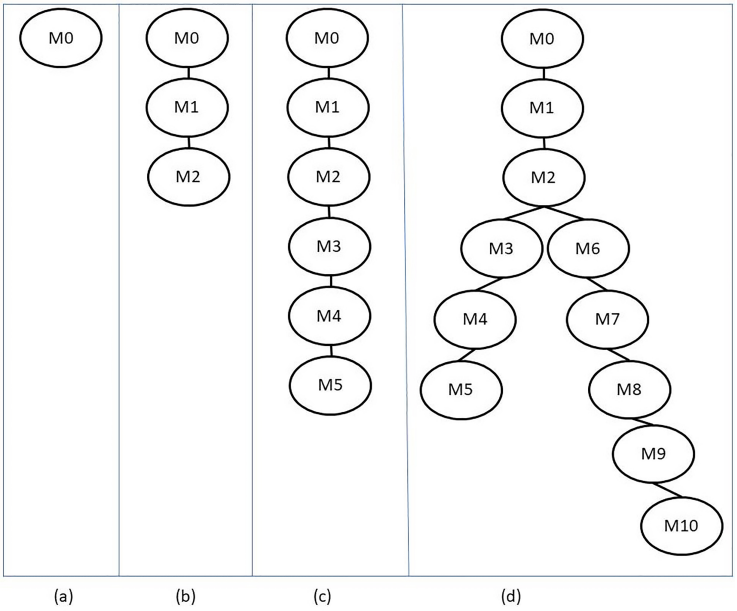


Figure 3. Evolution of the versioning tree while utilizing the flow of work on the stroke disease process model.

Given the action-state base-graph, the user may also wish to fuse (some of) the different states lying between action nodes A and B into one.

In this section, we provide different facilities and operations for summarizing state information and for facilitating the fusion of state nodes $\{S_1, \ldots, S_k\}$ *into a unique state node or into a certain number (less than k) of state nodes. We categorize such facilities/operations into four different types, applicable to sets* $\{S_1, \ldots, S_k\}$ *of states-nodes between the two action nodes A and B:*

- *Parameter operations, acting on the values of a specific parameter (*Section 6.1*);*
- *Filtering operations, to ignore one or more parameters (*Section 6.2*);*
- *Clustering operations, to partition the set* $\{S_1, \ldots, S_k\}$ *of states-nodes into subsets of state nodes, each to be fused into a unique state node (*Section 6.3*);*
- *Aggregation operations, aggregating the values of a given parameter taken from different state nodes (*Section 6.4*).*

*As we will see, such operations can be combined in different ways, for the sake of simplifying the action-state process-model (*see Section 6.5*). However, the basic idea is that filtering operations rule out non-interesting parameters; parameters operations simplify the values of a given parameter*

in each considered node; clustering fixes the state nodes to be fused (if no clustering is performed, $\{S_1, \ldots, S_k\}$ can be fused together into a unique state node), and once a set of state nodes to be fused is fixed, aggregation operators are applied to each parameter to aggregate the parameter values taken from such nodes.

In the following, we present such operations.

6.1. Parameter Operations

Parameter operations allow simplifying/summarizing the representation of a single parameter.

In some situations, experts may not be interested in raw parameter values but may prefer more abstract summarized information.

Specifically, as regards constant parameters, we support the use of discretization (see Section 5); when dealing with changing parameters, which assume the form of time series, different dimensionality reduction approaches are made available. All of these facilities are optional and can be applied, not only at the time of generalization over state nodes, but also as a pre-processing step and were therefore introduced in Section 5.

It is worth noting that parameter operations may produce, as a side effect, state coalescing; e.g., if all parameters assume the same values after discretization, they will be identical over two or more states. In this case, the corresponding state nodes are automatically fused.

Parameter operations modify the parameters tables associated with the state nodes at hand, which explicitly keep track of discretizations and dimensionality reductions.

6.2. Filtering Operations

An analyst, interacting with our system, may wish to restrict her/his analysis to a subset of parameters.

To this end, we provide an optional filtering facility, which allows the expert to check which parameters s/he wants to keep in the fused state node that will be generated. On the basis of domain knowledge, in fact, the expert may be aware that certain parameters are not particularly interesting for the problem at hand, but they might have been recorded in all traces as a routine. A graphical interface allows the user to select the parameters to be filtered out.

It is worth noting that filtering operations may produce, as a side effect, state coalescing; in fact, if some parameters have been filtered out, the remaining ones could be identical over two or more states. In this case, the corresponding state nodes are automatically fused.

Filtering operations modify the parameters tables associated with the state nodes at hand, since they eliminate some parameters.

6.3. Clustering Operations

An analyst may also want to cluster input states into more homogeneous groups, in order to build a more focused model. If this facility is not used, a single state node will be created through fusion, by generalizing over all the states between action nodes A and B in the input model; otherwise, fusion will be applied to each subset of state nodes (between A and B) obtained by clustering, to generate different state nodes, which will be drawn as parallel branches in the resulting process model. Clustering only operates on state information. Of course, there is a trade off between the richer information carried by the state separation and the greater readability of a model with a single state node. The interactivity of AS-SIM and the possibility of backtracking provide great flexibility in dealing with this issue.

This clustering operation is currently realized using the k-means algorithm [8]. Other algorithms may also be integrated in the future.

6.4. Aggregation Operations

Given a set of nodes (i.e., all the states nodes between two actions A and B, or all the nodes in one cluster), their fusion into a unique "summary" state node can be obtained by applying aggregation operations to each parameter in the state node description. Notably, different aggregation operations can be used for different parameters.

Non-discretized numeric constant parameters can be aggregated through resorting to the set of statistical techniques provided by AS-SIM. Namely, the user can calculate mean, variance, standard deviation, quartiles, median, and all the other basic statistical operators on numeric values.

Non-dimensionality-reduced changing parameters (i.e., time series) can be treated similarly: indeed, if all the time series samples have been kept, AS-SIM allows the user to align in time the different series belonging to different state nodes, cutting longer series in order to work with data of the same length. Missing data within a time series are completed by resorting to interpolation techniques. Once two or more time series are aligned sample by sample, AS-SIM allows the application of the statistical operators mentioned above on every single sample, generating, e.g., the time series of the mean values.

As regards discretized constant parameters, two strategies are possible. In the first case, the most frequent discretized value is used to substitute the set of discretized values belonging to the different state nodes. This aggregation strategy can be applied to numeric, symbolic, or categorical parameters.

In the case of numeric and categorical parameters, an alternative approach is also possible: each discretization interval/qualitative level is mapped to a number on an ordered scale; the statistical operations mentioned above are applied to such numbers for aggregation; then, the resulting number is mapped back onto the original domain.

As regards dimensionality-reduced time series numeric parameters, if a wavelet transform or a PAA technique has been adopted as a first step, the parameter is still represented as a sequence of values in time; simply, the number of these values is much lower than the number of the original samples. Therefore, all the aggregation operations working on a raw time series can still be applied. On the other hand, if temporal abstractions have been adopted, the time series will have been transformed into sequences of symbols, where every symbol corresponds to an interval of a fixed duration (set at the time of reducing dimensionality and specified in the parameter table) and qualitatively summarizes the behavior of the series in that interval. The different sequences of symbols (one per state) can be further collapsed into a single sequence, by selecting the most frequent symbol interval by interval. Alternatively, *complex* TAs [7] can be adopted. *Complex* TAs are temporal abstractions able to aggregate two series of intervals into a series of higher level intervals, in order to identify more complex patterns (e.g., an interval of an increasing trend followed by an interval of a decreasing trend, thus determining a peak). Such a method can be used when the original time series parameters have already been abstracted by means of basic TAs in the pre-processing phase. The latter two approaches can also be adopted on changing parameters that are symbolic or categorical in nature and can be therefore expressed as time series of qualitative values.

Aggregation operations modify the parameter table associated with the unique state node that is generated, since they put together all the parameters of the original state nodes they fuse.

6.5. Combining the Basic Operations

Different possibilities for combining the basic operations illustrated above are also provided.

Specifically, parameter operations can be optionally adopted as a first step, either during pre-processing or in the fusion of state nodes or both. Then, filtering and partitioning operations can be optionally applied.

The final step is the application of an appropriate aggregation operation, working either on raw parameters or on discretized/dimensionality reduced ones, as already explained in Section 6.4.

Example 3. *Reverting back to the example described in Section 5, we describe the work flow a user performs to manage the initial complexity of the action-state base-graph in Figure 2 and corresponding to the node M0 of the versioning tree in Figure 3a, to obtain a more manageable version using the parameter and state node operators described above. We focus on a particular section of this model, which is shown in Figure 4. In the latter, after admission to the SU and evaluation by a neurologist (the first action of this process excerpt), patients are treated with TPA to dissolve the thrombus obstructing the blood flow to the brain. Before and after this treatment, the patient's vital parameters are checked through continuous monitoring (see the state nodes (rectangles) in Figure 4). Before the therapy, monitoring is performed to guarantee the administration of the TPA under controlled conditions, while after the therapy, it is necessary to monitor the evolution of the patient's conditions to guide the next steps to be taken. After post-TPA monitoring, the installation of an ECG device for continuous cardiac monitoring may suggest the need for further investigations of patients who experience post-TPA issues. The model is obtained after executing the build-tree algorithm, which corresponds to step 4 of the tool workflow in Figure 1, which also includes the simple post-processing step that transforms the log-tree into a base-graph (see Section 6). Then, the flow of work moves to Steps 5 and 6, where the user can evaluate the current model and decide if the latter fulfills her/his requirements, terminating the procedure (Step 8 of the workflow), or it needs more elaboration (Step 7 and re-evaluation). In this case, further processing is needed and the flow of work continues with the steps below.*

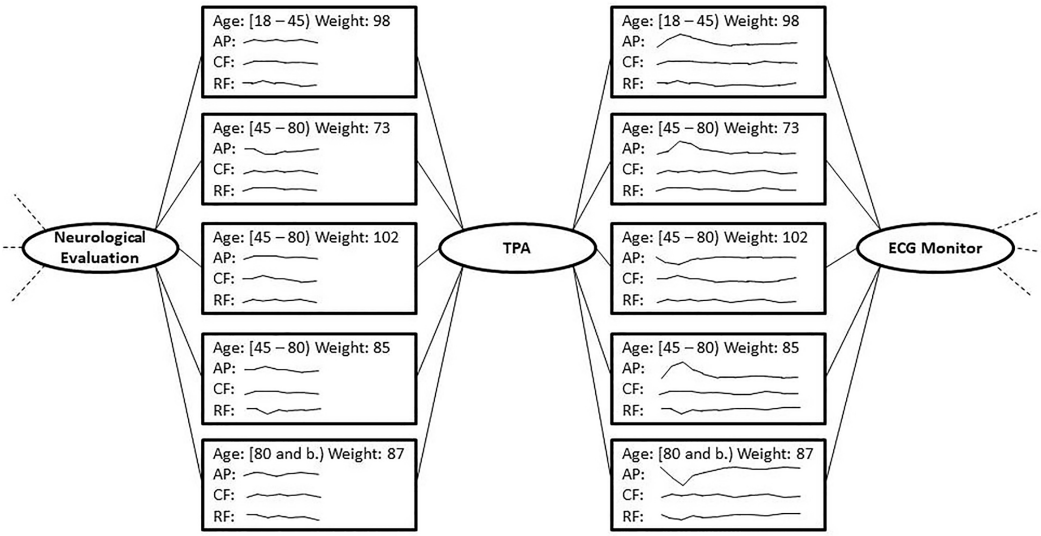


Figure 4. Excerpt of the action-state base-graph focusing on TPA administration.

As a first consideration, made in Step 6 of the tool workflow, the states between neurological evaluation and TPA are all related to patients eligible for TPA administration. Therefore, there is no need to differentiate patients with different conditions, since all of them are equally fit for receiving the thrombolysis treatment. In order to simplify the state contents, it is possible to filter out features which in clinical practice can be considered marginally important with respect to the others in the current context. For example, the stability of the patients' conditions is mainly checked by monitoring AP and CF trends; therefore, the RF can be filtered out. This is done in Step 7 of the workflow, using the filtering operator on each of the states between neurological evaluation and TPA. The resulting model, not shown for brevity, corresponds to node M1 of the versioning tree in Figure 3b, made available to the user looping back to Step 5, for further evaluation in Step 6. Here, s/he can decide to fuse all the considered states into one, applying aggregation operations in Step 7. In particular, the mean operator is applied for age and weight, while a point-to-point average

operator is applied for the time series present in the fused states. The result of the fusion can be seen in the part of model in Figure 5, corresponding to node M2 of the versioning tree in Figure 3b. The workflow starts again in Step 6 of the workflow in Figure 1, with evaluation of the latter model.

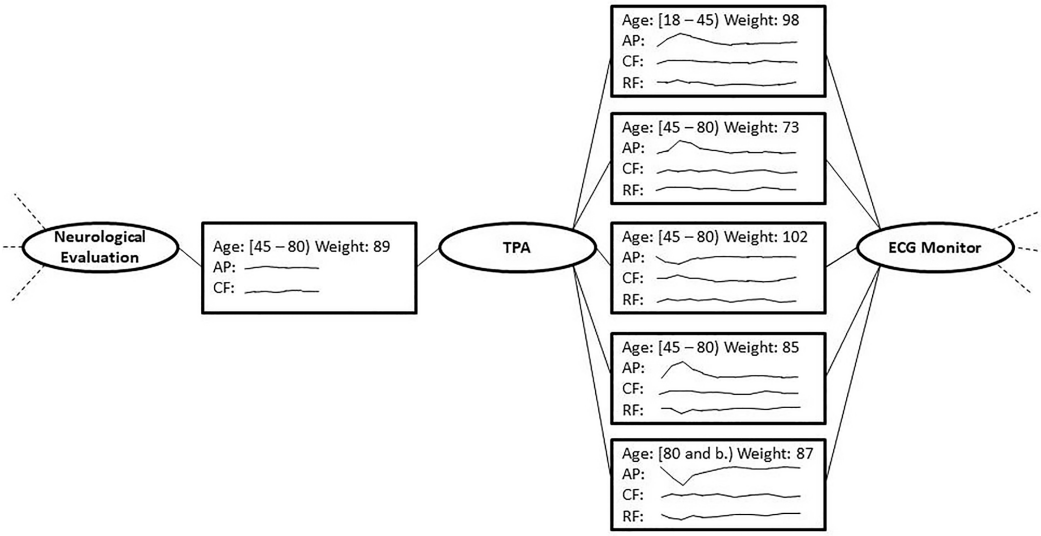


Figure 5. Excerpt of the action-state base-graph after the fusion of the states between neurological evaluation and TPA.

Here, we suppose that the user decides to further reduce the complexity by fusing the states between TPA and ECG monitor installation. As in the previous strategy, the user decides to filter out the RF feature and to simply fuse all the states into one. Step 7 of the workflow can be used to filter out RF, obtaining a model corresponding to the node M3 of the versioning tree (Figure 3c) and starting again from Step 6. Before fusing the considered states, it is worth noting that the vital sign monitoring after TPA lasts much longer than in the previous situation. Therefore, the user decides to apply a dimensionality reduction technique to the remaining time series (AP and CF) before performing the fusion. The flow of work moves to Step 7, in order to apply the TA operator to the AP and CF series of each state, with the aim of highlighting the trends of their evolution. The obtained model is related to node M4 in the versioning tree (Figure 3c) and the flow of work restarts from Step 6. The models related to nodes M3 and M4 are not shown for brevity. Finally, the user fuses all the considered states, moving to Step 7 to use the median operator on age and weight, and to resort to complex TAs for trends on AP and CF. Based on the medical knowledge, an increasing trend and a decreasing trend are aggregated into a stationary trend. The final result of this cycle of operations is the model shown in Figure 6, related to the node M5 in the versioning tree (Figure 3c).

Evaluating the model in Figure 6 during step 6 of the workflow, it appears that AP is aggregated as a stationary trend for the whole duration of monitoring. However, this does not match any of the situations described in the original states, where the AP of each patient has an unstable and never stationary trend. This fact should suggest problematic situations related to patients facing issues that could be investigated. The described fusion, instead, completely hides these problems and prevents the physicians from further monitoring these patients for cardiac problems after undergoing TPA.

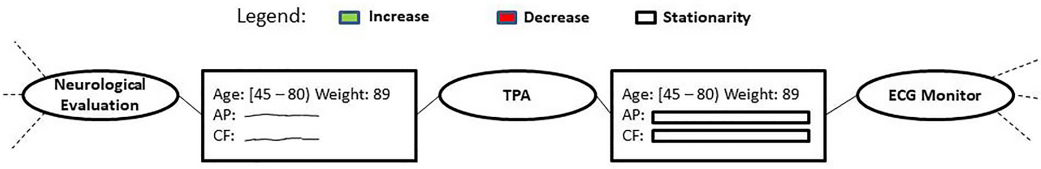


Figure 6. Excerpt of the action-state base-graph after fusion of all states between the TPA and ECG monitor.

Considering the issues above, the user decides to try a different strategy, in order to maintain the patients' peculiarities in this context. S/he uses the versioning tree in order to backtrack to the previous situation, selecting node M2, related to the process model in Figure 5, to start again. As in the previous cases, the user first uses Step 7 of the workflow to filter out RF from each considered state. This operation leads to a model related to the node M6 of the versioning tree (Figure 3d), spanning from M2 in a new branch, evolving in parallel to the nodes M3–M5, which ended with a refused model. The flow of operations restarts from Step 6, where the user decides to continue the procedure applying further operators. S/he then moves again to Step 7 to abstract PA using the trend TA operator and leaving CF unchanged, in order to be free to decide how to treat it after the state aggregation. Node M7 is thus created in the versioning tree, allowing the user to evaluate the related model in Step 6. Here, in order to identify sub-groups of homogeneous states sharing particular trend distributions, the user resorts to the clustering operator provided in Step 7. The clustering algorithm divides the considered states into two distinct groups: A and B. These groups are highlighted in the model related to node M8 of the versioning tree (see Figure 3d). The models related to nodes M7 and M8 of the versioning tree are not shown, for brevity. The user, positioned at Step 6 of the workflow, applies the aggregation operators offered by Step 7 to fuse all the states belonging to cluster A into a single state and performs the same operation to fuse all the states of cluster B into one. In both cases, the chosen aggregation operators are the mean for age and weight, the complex abstractions of the abstracted trends for AP and the point-to-point average for CF. The resulting model, containing the two obtained fused states, is shown in Figure 7 and relates to node M9 of the versioning tree (see Figure 3d).

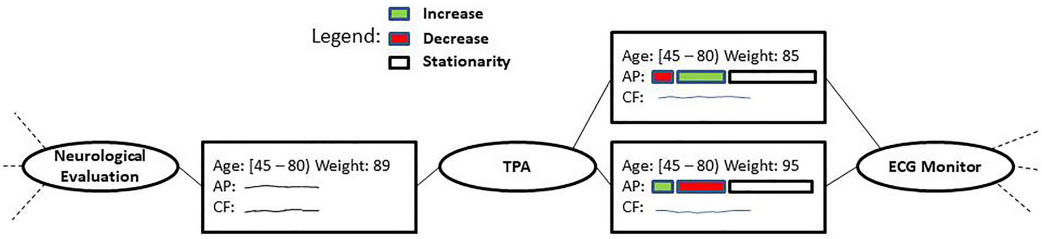


Figure 7. Excerpt of the action-state base-graph after fusion of the states between the TPA and ECG monitor after clustering.

Comparing the two fused states between the TPA and ECG monitor in Step 6, an unstable behavior of AP for the entire post-TPA monitoring can be observed for both, but with two very different evolutions of the relative trends. These differences could be useful for physicians, for a more in-depth analysis. As a last step, considering the prolonged monitoring time of the CF parameter, the user considers a dimensionality reduction strategy for it, using Step 7 for the last time to perform a PAA. This model, related to node M10 of the versioning tree (Figure 3d) and whose general shape is shown in Figure 8, after a final evaluation in Step 6, is considered satisfactory by the user; therefore, s/he can select this model as the final result and exit the procedure following Step 8 of the workflow.

Thanks to the techniques used above, the user can simplify the complete model shown in Figure 2 by appropriately addressing each context of the model. At the end of this work, it is possible to obtain a more general and easily interpretable model, such as the one shown in Figure 8.

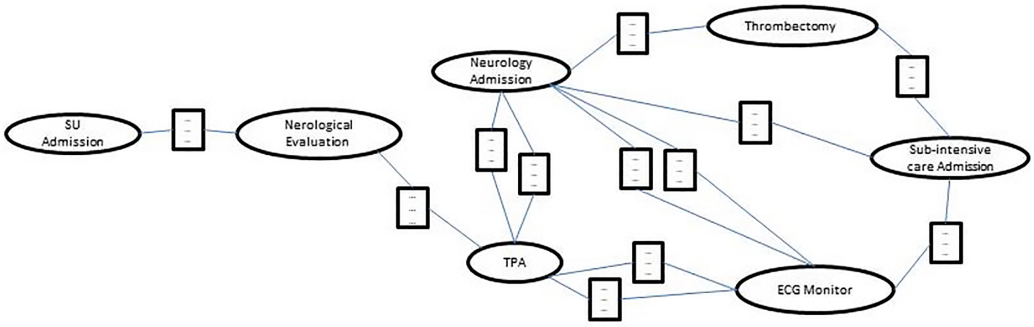


Figure 8. Action-state base-graph for thrombolysis treatment in stroke disease after state processing and fusion.

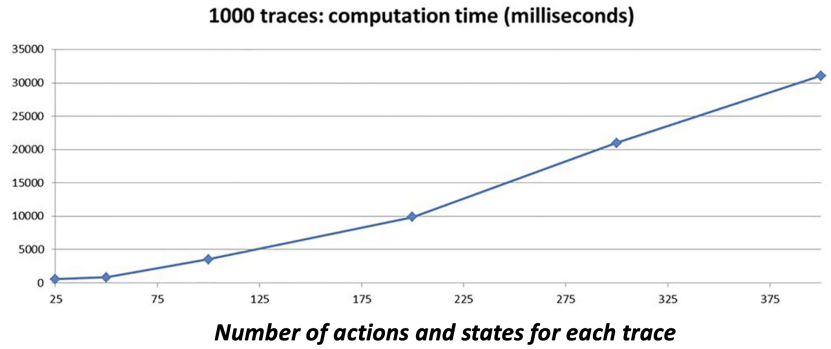
7. Quantitative Evaluation

In order to assess the system's performance, we conducted experiments to measure the time needed to mine the initial action-state process model from the input traces. This operation, corresponding to step 4 of the tool workflow in Figure 1 (build-tree algorithm) is the main computational task conducted by AS-SIM. The subsequent steps have less computational impact and their enactment and order depend on the user's choices. The evaluation was conducted by running the mining algorithm on a set of logs artificially obtained by randomly replicating parts of the traces in the stroke-related log presented in Sections 5 and 6. In particular, we prepared 6 sets, containing 100, 200, 500, 1000, 2500, and 5000 traces, respectively. For each dimension, we generated 6 logs containing sequences of 25, 50, 100, 200, 300, and 400 actions and states, for a total of 36 logs to be fed to the build-tree algorithm for testing.

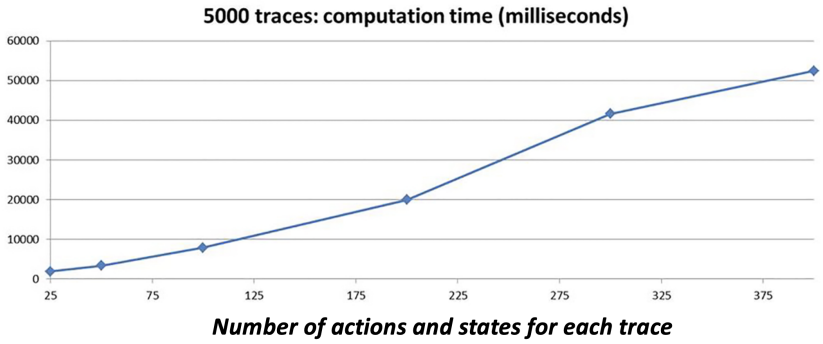
We performed these tests on a laptop equipped with an Intel Core I7 7700HQ CPU running at 2.80 GHz, 16 GB of DDR4 RAM, and Windows 10 operating system. The results are shown in Figures 9–11.

Figure 9 shows charts for performance of the build-tree algorithm (time in milliseconds) when fed logs containing the same amount of traces but with an increasing number of events and states for each trace. In particular, we show for brevity the performance obtained using logs containing 1000 (Figure 9a) and 5000 traces (Figure 9b), since the other charts show the same general shape. Analyzing these charts, it can be noted that the algorithm has a good scalability level, since the computation time seemed to increase with a nearly constant slope.

The same level of scalability is confirmed by the charts in Figure 10, which show the computation time in milliseconds needed to build the initial model from logs containing traces with the same dimensions but with an increasing number of traces. For brevity, we show the computational time where traces contained sequences of 100 (Figure 10a) and 400 actions and states (Figure 10b). The other charts show the same general shape.

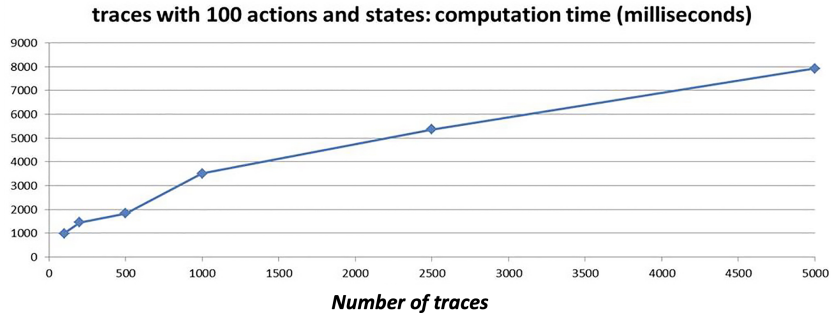


(**a**) Performance of the mining algorithm using logs with 1000 traces

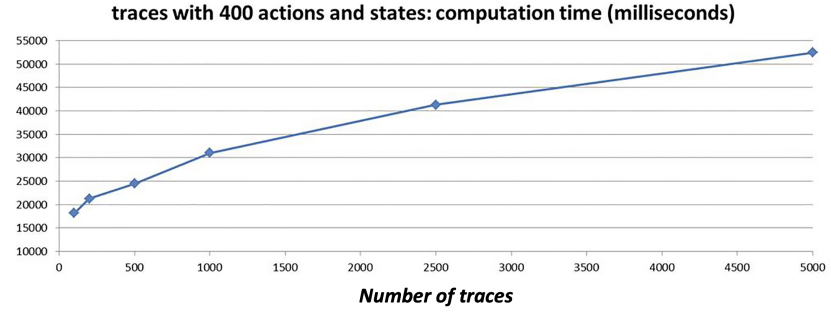


(**b**) Performance of the mining algorithm using logs with 5000 traces

Figure 9. Performance of the mining algorithm with respect to the increasing total number of actions and states in the traces.



(**a**) Performance of the mining algorithm using 100 actions and states in traces



(**b**) Performance of the mining algorithm using 400 actions and states in traces

Figure 10. Performance of the mining algorithm with respect to the increasing number of traces in the log.

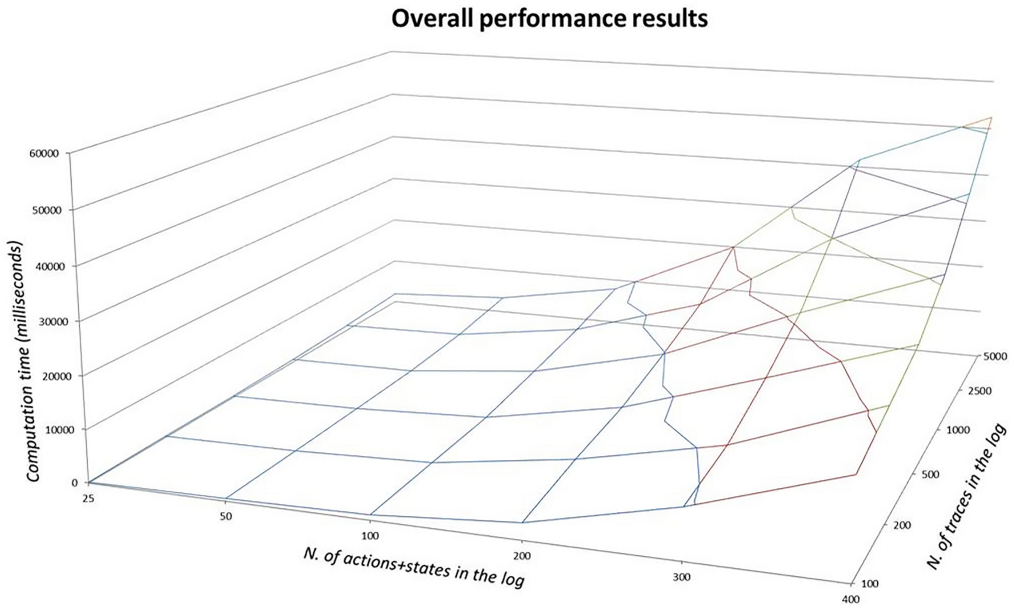


Figure 11. Overall performance of the mining algorithm as the number of traces in the log and the number of the total states and actions in the traces were varied.

The complete set of results is summarized in Figure 11, which shows the overall algorithm performance for each of the logs prepared for this evaluation. In particular, the reader can evaluate the variations of the computational time for logs containing the same amount of traces, but with increasing trace dimensions, by reading the chart in the horizontal direction. Following the vertical direction, it is possible to evaluate the performance when the traces had the same dimensions but their quantity increased.

Reading the chart in Figure 11, it can be noted that the scalability levels of our build-tree algorithm can be considered good in both directions, confirming that our approach can be seen, from a computational point of view, as a feasible choice for any real-life scenarios that can be taken into account.

8. Related Work

Process mining is nowadays a mature area of research, where many approaches have been proposed. As regards process model discovery in particular, back in 2016, professor van der Aalst was able to describe a rich family of methods and tools, which he categorized in his book [1] according to different dimensions. The dimensions can be summarized as follows:

- *representational bias*: this refers to the complexity of the process models that can be discovered using an approach. In particular, some algorithms are unable to mine and represent rather complex constructs, such as concurrency or arbitrary loops (see, e.g., the alpha miner [9]), while others are more expressive (for instance, a fuzzy miner [10] is able to learn hierarchical models); interestingly, the work in [11] describes a framework for selecting the most suitable miner for a given problem according to this dimension;
- *ability to deal with noise*: according to this dimension, the more mature algorithms are able to abstract from exceptional or infrequent behavior that may be recorded in the log; a heuristic miner [12], for instance, which extends the already mentioned alpha miner [9], has a good ability to deal with noise and has therefore also been adopted in medical domains [13];
- *completeness*: algorithms with a strong completeness assume that the log contains (almost) all the possible behaviors/traces (as the alpha miner [9] does, at least to some extent) and may encounter overfitting problems; a weak completeness may lead to

underfitting instead. A proper balance between overfitting and underfitting should be looked for, as in the approach in [14];

- *technique being used*: according to this dimension, we can distinguish between
  1. a direct approach, where the algorithm extracts a footprint from the log and directly exploits it to build a process model, as is the case of the alpha miner [9] and of the heuristic miner [12];
  2. a two-phase approach, where a first low-level model is initially built, and a high-level model is later derived from it: this is the case of the work in [14], which first builds a transition system, and then derives a more expressive model from it; to some extent, this is also the case of the original version of SIM [3];
  3. the divide-and-conquer approach, in which the log is split into smaller sub-logs, where mining can be easier, as in the case of the inductive miner [15];
  4. the computational intelligence approach, which adopts computational intelligence techniques, such as fuzzy sets (as in the fuzzy miner [10]) or genetic programming (as in the genetic miner [16]);
  5. the partial approach, which does not aim to learn a complete model, but focuses on rules or frequent patterns (as in [17]).

However, none of the classical algorithms mentioned by van der Aalst in his book deal with states. Moreover, classical approaches typically do not make use of domain knowledge and do not allow for interactivity with the user.

Indeed, only a few prior works have addressed the explicit representation of the flow of both actions and states in process model discovery.

In particular, in the area of medical data mining, the authors of [18,19] tackled the problem of medical knowledge formalization and proposed an approach to derive (manually or automatically) a model from medical data representing sequences of patient visits. Each visit determines the state of a patient and is followed by a prescription of therapies to be implemented until the next visit. From these input medical data, the authors extract a graph where nodes represent patient' states, and arcs (describing the evolution between pairs of states) are labeled using the therapies leading from the input to the output state.

Data-flow discovery was considered in [20,21], to identify decision points or to learn the rules that govern the decisions themselves. The authors in [20,21] adopted methods from the area of conformance checking to align the event log (that incorporates data measurements) with the process model and to generate a decision tree for every decision point, in order to select the correct alternative on the basis of the data. 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 based on the data values).

Unlike our approach, however, none of the above works tackled the general problem of discovering a process model that explicitly learns action and state nodes from input traces that contain both action and state information.

On the other hand, approaches that leverage domain knowledge in process model discovery have recently appeared in the process mining literature [22], to improve the quality of the discovered process itself.

For instance, the work in [23] presented an automated process discovery algorithm that exploits prior knowledge in the form of augmented information control nets, using ideas from Bayesian statistics. In [24], the authors focused on declarative process model discovery and proposed various automated techniques to remove redundant or non-interesting constraints from the discovered process models, as well as to guide the discovery approach by leveraging domain knowledge to identify the most relevant constraints from the domain point of view. The work in [25] presented an automated hybrid approach to process model discovery, where domain knowledge is encoded in the form of precedence constraints (over the topology of the resulting process models) and mining algorithms are formulated in terms of reasoning problems over such precedence constraints. It is, however, worth noting that, in all these contributions, the domain knowledge must be expressed in the form of

rules or constraints and the user cannot control the discovery of the process, which, in the end, can be very complex.

Recently, new process model discovery approaches have also begun emerging, which enable users to interactively integrate domain knowledge into the process model under construction. In particular, an interesting example is the interactive process discovery (IPD) presented in [26], which incrementally combines domain knowledge with the information in the event log, so as to improve the accuracy and understandability of the discovered process, while preserving its soundness. Specifically, starting from an initial process model, the user incrementally extends it by adding new elements and obtains feedback from IPD about the positioning of new actions and relations between actions. IDP uses "synthesized nets" (a type of Petri nets) as the underlying formalism for process models, and "synthesized rules" as a way to extend a process model into a new one by adding a transition and/or a place selected by the user at a certain time. The suitability and effectiveness of IPD for modeling real-world processes were assessed in [27], where the authors presented an experimental evaluation based on a case study in healthcare. The authors of [28] proposed an interactive process mining approach to discover dynamic risk models for patients suffering from chronic diseases based on patients' dynamic behavior provided by health sensor data. This approach is based on the interactive pattern recognition framework and allows a domain expert to adjust the model under construction interactively and iteratively. The obtained model can be leveraged to customize treatments based on a patient's unique behavior. Finally, in [29,30], the authors proposed an interactive process model discovery approach (and a tool) that allows users to learn a process model by incrementally adding observed process behavior (through new traces) to the process model under construction. A possible drawback of such an approach is the complexity of selecting the right traces when the noise in the data is high.

It is important to note that none of the mentioned works explicitly represent the flow of both actions and states discovered from the traces, while at the same time leveraging domain knowledge and/or interacting with the user to build more understandable models. In our opinion, such a comprehensive approach would be very useful in all domains where the state preconditions and/or the effects of actions on states have to be analyzed, general knowledge is available, and where experts can fruitfully interact with the tool, in order to obtain a more correct result. Therefore, we think that our proposal could become the starting point of a new line of research in the area of process model discovery.

As a final consideration, it is worth mentioning that process model discovery is not the only task of process mining. Conformance checking or anomaly detection (see, e.g., Ref. [31]), for instance, are attracting growing interest. At the moment, AS-SIM focuses on process model discovery, but in the future, conformance checking will be considered, trying to leverage state information as well. In particular, when considering medical applications, conformance checking allows experts to compare the actual process model implemented at a given organization to the golden standard one, i.e., the clinical guideline. Such a comparison would help to identify bottlenecks and limitations of the local process and constitutes a fundamental step towards an improvement in patient care. We believe that the exploitation of state information could provide a more accurate comparison and thus greater help towards an increase in patient management quality.

Last but not least, it is important to note that, in process discovery, the action workflow perspective is not the only existing point of view: some works (e.g., Ref. [32]) focus on the resource perspective; these works are, however, loosely related to our contribution.

9. Future Work and Conclusions

In this paper, we have proposed AS-SIM, the first process model discovery approach that mines and provides an explicit representation of the flow of both actions and states from input traces.

As a future work, we plan to conduct an extensive experimental evaluation, thus enriching the initial experimental results we described in this paper. First, we plan to

work on a large collection of real medical traces, suitably derived from the freely-available MIMIC clinical database [33]. Moreover, and more importantly, we plan to conduct a real-world validation study in a hospital setting. Indeed, we are collaborating with the Integrated Laboratory of Artificial Intelligence and Medical Informatics, a shared entity between the Computer Science Institute of our University and the Hospital of Alessandria, Italy. Within this collaboration, it will be possible to set up a wide-ranging validation study, involving different hospital departments, different medical experts, and multiple patients. We believe that such a study will provide significant feedback about our innovative and interactive process mining approach.

From a more methodological viewpoint, we aim to extend AS-SIM with a broad class of *merge* and *abstraction* operations, which will allow experts to move progressively from an action-state log-tree (which is a potentially overfitting process model) to a increasingly generalized graph-based process model. In particular, we will work on the definition of *merge* and *abstraction* operations, which generalize and complement the current operations in SIM, so as to also consider state nodes.

After the implementation of all these operations is realized, making the process discovery task complete, we will consider an additional process mining task, namely conformance checking. Indeed, while process discovery is the basic fundamental step of our approach, its realization can be seen as the starting point for reaching additional objectives. Conformance checking aims to compare the actual process model mined from process traces to an ideal (already available) process model; we will also study novel approaches to conformance checking able to leverage the state information, making this comparison more accurate. This research direction seems to be particularly promising, e.g., in medicine; in this case, the actual process mined at a given hospital can be compared to a gold standard process, i.e., a clinical guideline. A more complete and reliable comparison will allow experts to better identify bottlenecks and limitations of the local process, and will provide fundamental support towards an improvement in patient care.

Author Contributions: Conceptualization, All authors; methodology, P.T., S.M., G.L. and A.B.; software, M.G., M.S. and A.B.; validation, G.L.; formal analysis, P.T. and A.B.; data curation, M.S.; writing—original draft preparation, S.M., P.T. and G.L.; writing—review and editing, All authors; supervision, P.T. and S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is unavailable due to privacy or ethical restrictions.

Acknowledgments: This research has been partially supported by the INDAM—GNCS Project 2023.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. van der Aalst, W.M.P. *Process Mining—Data Science in Action*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2016. [CrossRef]
2. Wang, S.; McDermott, M.B.A.; Chauhan, G.; Hughes, M.C.; Naumann, T.; Ghassemi, M. MIMIC-Extract: A Data Extraction, Preprocessing, and Representation Pipeline for MIMIC-III. In Proceedings of the ACM Conference on Health, Inference, and Learning, Toronto, ON, Canada, 2–4 April 2020.
3. Bottrighi, A.; Canensi, L.; Leonardi, G.; Montani, S.; Terenziani, P. Interactive mining and retrieval from process traces. *Expert Syst. Appl.* **2018**, *110*, 62–79. [CrossRef]
4. Bottrighi, A.; Guazzone, M.; Leonardi, G.; Montani, S.; Striani, M.; Terenziani, P. AS-SIM: An Approach to Action-State Process Model Discovery. In Proceedings of the Foundations of Intelligent Systems—26th International Symposium, ISMIS 2022, Cosenza, Italy, 3–5 October 2022; Ceci, M., Flesca, S., Masciari, E., Manco, G., Ras, Z.W., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2022; Volume 13515, pp. 336–345. [CrossRef]
5. Buijs, J.C.A.M.; van Dongen, B.F.; van der Aalst, W.M.P. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In Proceedings of the On the Move to Meaningful Internet Systems: OTM 2012, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, 10–14 September 2012; Proceedings, Part I; Meersman, R., Panetto, H., Dillon, T.S., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7565, pp. 305–322. [CrossRef]

6. Keogh, E.; Chakrabarti, K.; Pazzani, M.; Mehrotra, S. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.* **2000**, *3*, 263–286. [CrossRef]
7. Bellazzi, R.; Larizza, C.; Riva, A. Temporal Abstractions for Interpreting Diabetic Patients Monitoring Data. *Intell. Data Anal.* **1998**, *2*, 97–122. [CrossRef]
8. Hartigan, J.A. *Clustering Algorithms*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1975.
9. van der Aalst, W.M.P.; van Dongen, B.F. Discovering Workflow Performance Models from Timed Logs. In Proceedings of the Engineering and Deployment of Cooperative Information Systems, First International Conference, EDCIS 2002, Beijing, China, 17–20 September 2002; Han, Y., Tai, S., Wikarski, D., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2480, pp. 45–63. [CrossRef]
10. Günther, C.W.; van der Aalst, W.M.P. Fuzzy Mining—Adaptive Process Simplification Based on Multi-perspective Metrics. In Proceedings of the Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, 24–28 September 2007; Alonso, G., Dadam, P., Rosemann, M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4714, pp. 328–343. [CrossRef]
11. Al-Absi, M.A.; R'bigui, H. Process Discovery Techniques Recommendation Framework. *Electronics* **2023**, *12*, 3108. [CrossRef]
12. Weijters, A.; Aalst, W.; Medeiros, A. *Process Mining with the Heuristics Miner-Algorithm*; Cirp Annals-Manufacturing Technology; Elsevier: New York, NY, USA, 2006; Volume 166.
13. Montani, S.; Leonardi, G.; Striani, M.; Quaglini, S.; Cavallini, A. Multi-level abstraction for trace comparison and process discovery. *Expert Syst. Appl.* **2017**, *81*, 398–409. [CrossRef]
14. van der Aalst, W.M.P.; Rubin, V.; Verbeek, H.M.W.; van Dongen, B.F.; Kindler, E.; Günther, C.W. Process mining: A two-step approach to balance between underfitting and overfitting. *Softw. Syst. Model.* **2010**, *9*, 87–111. [CrossRef]
15. Leemans, S.J.J.; Fahland, D.; van der Aalst, W.M.P. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In Proceedings of the Business Process Management Workshops—BPM 2013 International Workshops, Beijing, China, 26 August 2013; Revised Papers; Lohmann, N., Song, M., Wohed, P., Eds.; Lecture Notes in Business Information Processing; Springer: Berlin/Heidelberg, Germany, 2013; Volume 171, pp. 66–78. [CrossRef]
16. Bratosin, C.; Sidorova, N.; van der Aalst, W.M.P. Discovering Process Models with Genetic Algorithms Using Sampling. In Proceedings of the Knowledge-Based and Intelligent Information and Engineering Systems—14th International Conference, KES 2010, Cardiff, UK, 8–10 September 2010; Proceedings, Part I; Setchi, R., Jordanov, I., Howlett, R.J., Jain, L.C., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6276, pp. 41–50. [CrossRef]
17. Maggi, F.M.; Dumas, M.; García-Bañuelos, L.; Montali, M. Discovering Data-Aware Declarative Process Models from Event Logs. In Proceedings of the Business Process Management—11th International Conference, BPM 2013, Beijing, China, 26–30 August 2013; Daniel, F., Wang, J., Weber, B., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8094, pp. 81–96.
18. Kamisalic, A.; Riaño, D.; Welzer, T. Formalization and acquisition of temporal knowledge for decision support in medical processes. *Comput. Methods Programs Biomed.* **2018**, *158*, 207–228. [CrossRef] [PubMed]
19. Kamisalic, A.; Riaño, D.; Kert, S.; Welzer, T.; Zlatolas, L.N. Multi-level medical knowledge formalization to support medical practice for chronic diseases. *Data Knowl. Eng.* **2019**, *119*, 36–57. [CrossRef]
20. de Leoni, M.; Felli, P.; Montali, M. A Holistic Approach for Soundness Verification of Decision-Aware Process Models. In Proceedings of the Conceptual Modeling—37th International Conference, ER 2018, Xi'an, China, 22–25 October 2018; Trujillo, J., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2018; Volume 11157, pp. 219–235. [CrossRef]
21. de Leoni, M.; van der Aalst, W.M.P. Data-aware process mining: Discovering decisions in processes using alignments. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, 18–22 March 2013; Shin, S.Y., Maldonado, J.C., Eds.; ACM: New York, NY, USA, 2013; pp. 1454–1461. [CrossRef]
22. Schuster, D.; van Zelst, S.J.; van der Aalst, W.M.P. Utilizing domain knowledge in data-driven process discovery: A literature review. *Comput. Ind.* **2022**, *137*, 103612. [CrossRef]
23. Rembert, A.J.; Omokpo, A.; Mazzoleni, P.; Goodwin, R.T. Process Discovery Using Prior Knowledge. In Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013), Berlin, Germany, 2–5 December 2013; Basu, S., Pautasso, C., Zhang, L., Fu, X., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 328–342. [CrossRef]
24. Maggi, F.M.; Bose, R.P.J.C.; van der Aalst, W.M.P. A Knowledge-Based Integrated Approach for Discovering and Repairing Declare Maps. In Proceedings of the Proceedings 25th International Conference on Advanced Information Systems Engineering (CAiSE 2013), Valencia, Spain, 17–21 June 2013; Salinesi, C., Norrie, M.C., Pastor, Ó., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 433–448. [CrossRef]
25. Greco, G.; Guzzo, A.; Lupia, F.; Pontieri, L. Process Discovery under Precedence Constraints. *ACM Trans. Knowl. Discov. Data* **2015**, *9*, 1–39. [CrossRef]
26. Dixit, P.M.; Verbeek, H.M.W.; Buijs, J.C.A.M.; van der Aalst, W.M.P. Interactive Data-Driven Process Model Construction. In Proceedings of the Conceptual Modeling—37th International Conference, ER 2018, Xi'an, China, 22–25 October 2018; Trujillo, J., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2018; Volume 11157, pp. 251–265. [CrossRef]

27. Benevento, E.; Aloini, D.; van der Aalst, W.M.P. How Can Interactive Process Discovery Address Data Quality Issues in Real Business Settings? Evidence from a Case Study in Healthcare. *J. Biomed. Inform.* **2022**, *130*, 104083. [CrossRef] [PubMed]
28. Valero-Ramon, Z.; Fernandez-Llatas, C.; Valdivieso, B.; Traver, V. Dynamic Models Supporting Personalised Chronic Disease Management through Healthcare Sensors with Interactive Process Mining. *Sensors* **2020**, *20*, 5330. [CrossRef] [PubMed]
29. Schuster, D.; van Zelst, S.J.; van der Aalst, W.M.P. Incremental Discovery of Hierarchical Process Models. In Proceedings of the Research Challenges in Information Science, Limassol, Cyprus, 23–25 September 2020; Dalpiaz, F., Zdravkovic, J., Loucopoulos, P., Eds.; Springer: Cham, Switzerland, 2020; pp. 417–433. [CrossRef]
30. Schuster, D.; van Zelst, S.J.; van der Aalst, W.M. Cortado: A dedicated process mining tool for interactive process discovery. *SoftwareX* **2023**, *22*, 101373. [CrossRef]
31. Bin Ahmadon, M.A.; Yamaguchi, S. Verification Method for Accumulative Event Relation of Message Passing Behavior with Process Tree for IoT Systems. *Information* **2020**, *11*, 232. [CrossRef]
32. Utama, N.I.; Sutrisnowati, R.A.; Kamal, I.M.; Bae, H.; Park, Y.J. Mining Shift Work Operation from Event Logs. *Appl. Sci.* **2020**, *10*, 7202. [CrossRef]
33. Johnson, A.E.; Pollard, T.J.; Shen, L.; Lehman, L.w.H.; Feng, M.; Ghassemi, M.; Moody, B.; Szolovits, P.; Anthony Celi, L.; Mark, R.G. MIMIC-III, a freely accessible critical care database. *Sci. Data* **2016**, *3*, 160035. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.