algorithms

MDPI

Article

Efficient Time and Space Representation of Uncertain Event Data

Marco Pegoraro *, Merih Seran Uysal and Wil M. P. van der Aalst

Process and Data Science Group (PADS), Department of Computer Science, RWTH Aachen University, 52062 Aachen, Germany; uysal@pads.rwth-aachen.de (M.S.U.); wvdaalst@pads.rwth-aachen.de (W.M.P.v.d.A.)
* Correspondence: pegoraro@pads.rwth-aachen.de

Received: 30 September 2020; Accepted: 6 November 2020; Published: 9 November 2020

Abstract: Process mining is a discipline which concerns the analysis of execution data of operational processes, the extraction of models from event data, the measurement of the conformance between event data and normative models, and the enhancement of all aspects of processes. Most approaches assume that event data is accurately captured behavior. However, this is not realistic in many applications: data can contain uncertainty, generated from errors in recording, imprecise measurements, and other factors. Recently, new methods have been developed to analyze event data containing uncertainty; these techniques prominently rely on representing uncertain event data by means of graph-based models explicitly capturing uncertainty. In this paper, we introduce a new approach to efficiently calculate a graph representation of the behavior contained in an uncertain process trace. We present our novel algorithm, prove its asymptotic time complexity, and show experimental results that highlight order-of-magnitude performance improvements for the behavior graph construction.

Keywords: process mining; uncertain data; partial order

1. Introduction

The pervasive diffusion of digitization, which gained momentum thanks to advancements in electronics and computing at the end of the last century, has brought a wave of innovation in the tools supporting businesses and companies. Recent decades have seen the rise of *Process-Aware Information Systems* (PAISs)—useful to structurally support processes in a business—as well as research disciplines such as Business Process Management (BPM) and *process mining*.

Process mining [1] is a field of research that enables process analysis in a data-driven manner. Process mining analyses are based on recordings of tasks and events in a process, memorize in an ensemble of information systems which support business operations. These recordings are exported and systematically collected in databases called *event logs*. Using an event log as a starting point, process mining techniques can automatically obtain a process model illustrating the behavior of the real-life process (*process discovery*) and identify anomalies and deviations between the execution data of a process and a normative model (*conformance checking*). Process mining is a subfield of data science which is quickly growing in interest both in academia and industry. Over 30 commercial software tools are available on the market for analyzing processes and their execution data. Process mining tools are used by process experts to analyze processes in tens of thousands of organizations, e.g., within Siemens, over 6000 employees actively use process mining to improve internal procedures.

Commercial process mining software can discover and build a process model from an event log. Most of the process discovery algorithms implemented in these tools are based on tallying the number of *directly-follows relationships* between activities in the execution data of the process. The more frequently a specific activity immediately follows another one in the execution log of a process,

the stronger a causality and/or precedence implication between the two activities is understood to be. Such directly-follows relationships are also the basis for the identification of more complex and abstract constructs in the workflow of a process, such as interleaving or parallelism of activities. These relationships between activities are often represented in a labeled directed graph called the *Directly-Follows Graph* (DFG).

In recent times, a new type of event logs has gained research interest: *uncertain event logs* [2]. Such execution logs contain, rather than precise values, an indication of the possible values that event attributes can acquire. In this paper, we will consider the setting where uncertainty is represented by either an interval or a set of possible values for an event attribute. Moreover, we will consider the case in which an event has been recorded in the event log albeit it did not happen in reality.

Uncertainty in event logs is best illustrated with a real-life example of a process that can generate uncertain data in an information system. Let us consider the following process instance, a simplified version of anomalies that are actually occurring in processes of the healthcare domain. An elderly patient enrolls in a clinical trial for an experimental treatment against myeloproliferative neoplasms, a class of blood cancers. The enrollment in this trial includes a lab exam and a visit with a specialist; then, the treatment can begin. The lab exam, performed on the 8th of July, finds a low level of platelets in the blood of the patient, a condition known as thrombocytopenia (TP). At the visit, on the 10th of May, the patient self-reports an episode of night sweats on the night of the 5th of July, prior the lab exam: the medic notes this, but also hypothesized that it might not be a symptom, since it can be caused not by the condition but by external factors (such as very warm weather). The medic also reads the medical records of the patient and sees that shortly prior to the lab exam, the patient was undergoing a heparine treatment (a blood-thinning medication) to prevent blood clots. The thrombocytopenia found with the lab exam can then be primary (caused by the blood cancer) or secondary (caused by other factors, such as a drug). Finally, the medic finds an enlargement of the spleen in the patient (splenomegaly). It is unclear when this condition has developed: it might have appeared in any moment prior to that point. The medic decides to admit the patient in the clinical trial, starting 12th of July.

These events generate the trace of Table 1 in the information system of the hospital. For clarity, the timestamp field only reports the day of the month.

Table 1. The uncertain trace of an instance of healthcare process used as running example. The "Case ID" is a unique identifier for all events in a single process case; the "Event ID" is a unique identifier for the events in the trace. The "Timestamp" field indicates either the moment in time in which the event has happened, or the interval of time in which the event may have happened. The "Activity" field indicates the possible choices for the activity instantiated by the event. Lastly, the "Indeterminate event" field contains a "!" if the corresponding event has surely occurred, and a "?" if it might have been recorded despite not occurring in reality. For the sake of readability, in the timestamps column only reports the day of the month.

Case ID	Event ID	Timestamp	Activity	Indet. Event
ID327	e_1	5	*NightSweats*	?
ID327	e_2	8	{*PrTP*, *SecTP*}	!
ID327	e_3	[4, 10]	*Splenomeg*	!
ID327	e_4	12	*Adm*	!

Event e_2 has been recorded with two possible activity labels (*PrTP* or *SecTP*). This is an example of uncertainty on activities. Some events, e.g., e_3, do not have a precise timestamp but a time interval in which the event could have happened has been recorded: in some cases, this causes the loss of a precise ordering of events (e.g., e_2 and e_3). This is an instance of uncertainty on the time dimension, i.e., on timestamps. As evident by the "?" symbol, e_1 is an indeterminate event: it has been recorded, but it is not guaranteed to have actually happened. Conversely, the "!" symbol indicates that the

event has been recorded while certainly occurring in reality, i.e., it has been recorded correctly in the information system (e.g., the event e_4).

Quality problems and imprecision in data recording such as the ones described in the running example as source of uncertainty are not uncommon; in some settings, they are a frequent occurrence. Healthcare processes are specifically know to be afflicted by these sorts of data anomalies, especially if parts of the process rely on recording information on paper [3,4]. Existing process mining software cannot manage such uncertain event data. When mining the processes where uncertainty in execution data is prominent, a natural first approach is to filter the event log eliminating cases where uncertainty appear. Unfortunately, in processes with a large portion of cases are affected by such data anomalies, filtering without losing essential information about the process is not feasible.

Consequently, new process mining methods to inspect and analyze it must be developed. Uncertain timestamps are the most prominent and critical source of uncertain behavior in a process trace. For example, if n events have uncertain timestamps such that their order is unknown, the possible configurations that the control-flow of the trace can assume are all the $n!$ permutations of the events, in the case where all events in a case have timestamps defined by mutually overlapping intervals. This is the worst possible scenario in terms of amount of uncertain behavior introduced by uncertainty on the timestamps of the events ins a trace. Thus, it is important to capture the time relationships between events in a compact and effective way. This is accomplished by the construction of a *behavior graph*, a directed acyclic graph that expresses precedence between events. Figure 1 shows the behavior graph of the process trace in Table 1; every known precedence relationship between events is represented by the edges of the graph, while the pairs of event for which the order is unknown remain unconnected. Effectively, this creates a representation of the partial order where the arcs are defined by the possible values of the timestamps contained in the trace, and where the nodes may refer to sets of possible activities. As we will see, this construct is central to effectively implement both process discovery and conformance checking applied to uncertain event data.

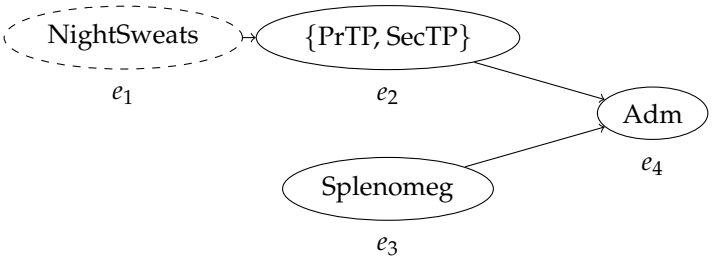


Figure 1. The behavior graph of the trace in Table 1. Every node represents an event; the labels in the nodes represent the activity, or set of activities, associated with the event. The arcs represent the partial order relationship between events as defined by their timestamps. The indeterminate event, which might not have occurred, is represented by a dashed node.

In a previous paper [5], we presented a time-effective algorithm for the construction of the behavior graph of an uncertain process trace, attaining quadratic time complexity on the number of events in the trace.

This paper elaborates on this previous result, by providing the proof of the correctness of the new algorithm. Additionally, we will show the improvement in performance both theoretically, via asymptotic complexity analysis, and in practice, with experiments on various uncertain event logs comparing computation times of the baseline method against the novel construction algorithm. Furthermore, the version of the algorithms presented in this paper is refined so to preprocess uncertain traces in linear time, individuating the variants—which share the same behavior graph–, and proceed to perform the construction of the behavior graph only once per variant. This slightly improves performance, and more importantly, enables the representation of an uncertain event log as a multiset of

behavior graphs, greatly reducing the memory requirements to store the log. This enables a streamlined application of process mining techniques on event data where uncertainty is present.

The algorithms have been implemented within the PROVED (*PRocess mining OVer uncErtain Data*) library (https://github.com/proved-py/proved-core/tree/Efficient_Time_and_Memory_Representation_for_Uncertain_Event_Data), based on the PM4Py process mining framework [6].

The reminder of the paper is structured as follows: Section 2 motivates the study of uncertainty in process mining by illustrating an example of conformance checking over uncertain event data. Section 3 strengthens the motivation showing the discovery of process models of uncertain event logs. Section 4 provides formal definitions, describes the baseline technique for our research, and shows a new and more efficient method to obtain a behavior graph of an uncertain trace. Section 5 presents the analysis of asymptotic complexity for both the baseline and the novel method. Section 6 shows results of experiments on both synthetic and real-life uncertain event logs comparing the efficiency of both methods to compute behavior graphs. Section 7 explores recent related works in the context of uncertain event data and the management of alterations of data in process mining. Finally, Section 8 discusses the output of the experiments and concludes the paper.

2. Conformance Checking over Uncertain Data

Conformance checking is one of the main tasks in process mining, and consists of measuring the deviation between process execution data (usually in the form of a trace) and a reference model. This is particularly useful for organization, since it enables them to compare historical process data against a normative model created by process experts to identify anomalies and deviations in their operations.

Let us assume that we have access to a normative model for the disease of the patient in the running example, shown in Figure 2.

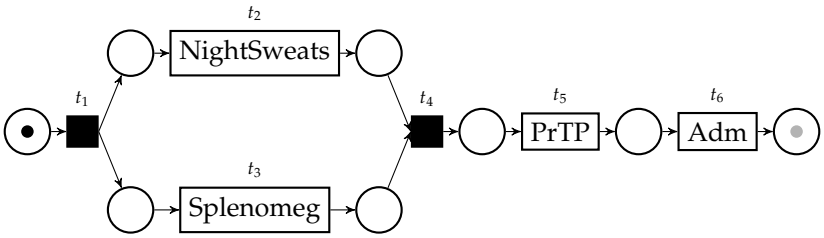


Figure 2. A normative model for the healthcare process case in the running example. The initial marking is displayed; the gray "token slot" represents the final marking.

This model essentially states that the disease is characterized by the occurrence of night sweats and splenomegaly on the patient, which can be verified concurrently, and then should be followed by primary thrombocytopenia. We would like to measure the conformance between the trace in Table 1 and this normative model. A very popular conformance checking technique works via the computation of *alignments* [7]. Through this technique, we are able to identify the deviations in the execution of a process, in the form of behavior happening in the model but not in the trace, and behavior happening in the trace but not in the model. These deviations are identified, and used as basis to compute a conformance score between the trace and the process model.

The formulation of alignments in [7] is not applicable to an uncertain trace. In fact, depending on the instantiation of the uncertain attributes of events—like the timestamp of e_3 in the trace—the order of event may differ, and so may the conformance score. However, we can look at the best- and worst-case scenarios: the instantiation of attributes of the trace that entails the minimum and maximum number of deviations with respect to the reference model. In our example, two possible outcomes for the sample trace are $\langle NightSweats, Splenomeg, PrTP, Adm\rangle$ and $\langle SecTP, Splenomeg, Adm\rangle$; both represent the sequence of event that might have happened in reality, but their conformance score is very different. The alignment of the first trace against the reference model can be seen in Table 2, while the alignment of the second trace can be seen in Table 3. These two outcomes of the uncertain

trace in Table 1 represent, respectively, the minimum and maximum amount of deviation possible with respect to the reference model, and define then a lower and upper bound for conformance score.

Table 2. An optimal alignment for $\langle NightSweats, Splenomeg, PrTP, Adm \rangle$, one of the possible instantiations of the trace in Table 1, against the model in Figure 2. This alignment has a deviation cost equal to 0, and corresponds to the best-case scenario for conformance between the process model and the uncertain trace.

$\gg$	**NightSweats**	**Splenomeg**	$\gg$	**PrTP**	**Adm**
τ	NightSweats	Splenomeg	τ	PrTP	Adm
t_1	t_2	t_3	t_4	t_5	t_6

Table 3. An optimal alignment for $\langle SecTP, Splenomeg, Adm \rangle$, one of the possible instantiations of the trace in Table 1, against the model in Figure 2. This alignment has a deviation cost equal to 3, caused by 2 moves on model and 1 move on log, and corresponds to the worst-case scenario for conformance between the process model and the uncertain trace.

$\gg$	**SecTP**	$\gg$	**Splenomeg**	$\gg$	$\gg$	**Adm**
τ	$\gg$	NightSweats	Splenomeg	τ	PrTP	Adm
t_1		t_2	t_3	t_4	t_5	t_6

The minimum and maximum bounds for conformance score of an uncertain trace and a reference process model can be found with the uncertain version of the alignment technique that we first described in [2]. To find such bounds, it is necessary to build a Petri net able to simulate all possible behaviors in the uncertain trace, called the *behavior net*. Obtaining a behavior net is possible through a construction that uses behavior graphs as a starting point, using the structural information therein contained to connect places and transitions in the net. The behavior net of the trace in Table 1 is shown in Figure 3.

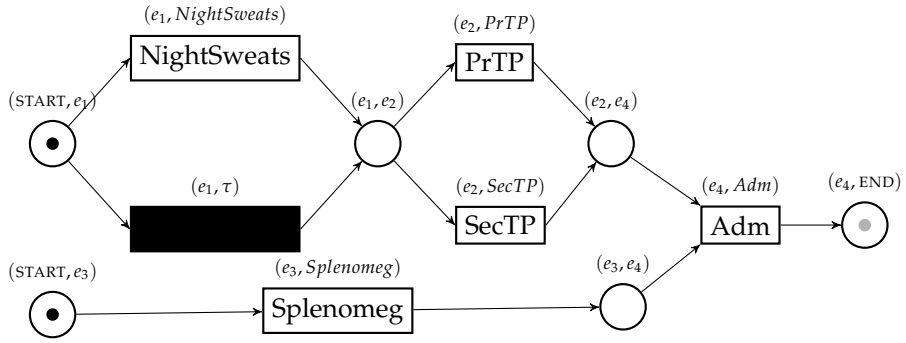


Figure 3. The behavior net representing the behavior of the uncertain trace in Table 1 and obtained thanks to its behavior graph. The initial marking is displayed; the gray "token slot" represents the final marking. This artifact is necessary to perform conformance checking between uncertain traces and a reference model.

The alignments in Tables 2 and 3 show how we can get actionable insights from process mining over uncertain data. In some applications it is reasonable and appropriate to remove uncertain data from an event log via filtering, and then compute log-level aggregate information—such as total number of deviations, or average deviations per trace—using the remaining certain data. Even in processes where this is possible, doing so prevents the important process mining task of case diagnostic. Conversely, uncertain alignments allow not only to have best- and worst-case scenarios for a trace, but also to individuate the specific deviations affecting both scenarios. For instance, the alignments of

the running example can be implemented in a system that warns the medics that the patient might have been affected by a secondary thrombocytopenia not explained by the model of the disease. Since the model indicates that the disease should develop primary thrombocytopenia as a symptom, this patient is at risk of both types of platelets deficit simultaneously, which is a serious situation. The medics can then intervene to avoid this complication, and performing more exams to ascertain the cause of the patient's thrombocytopenia.

3. Process Discovery over Uncertain Data

Process discovery is another main objective in process mining, and involves automatically creating a process model from event data. Many process discovery algorithms rely on the concept of *directly-follows relationships* between activities to gather clues on how to structure the process model. *Uncertain Directly-Follows Graphs* (UDFGs) enable the representation of directly-follows relationships in an event log under conditions of uncertainty in the event data; they consist of directed graphs where the activity labels appearing in an event log constitute the nodes, and the edges are decorated with information on the minimum and maximum frequency observable for the directly-follows relation between pair of activities.

Let us examine an example of UDFG. To build a significant example, we need to introduce an entire uncertain event log; since the full table notation for uncertain traces becomes cumbersome for entire logs, let us use a shorthand simplified notation. In a trace, we represent an uncertain event with multiple possible activity labels by listing all the associated labels between curly braces.

When two events have mutually overlapping timestamps, we write their activity labels between square brackets, and we indicate indeterminate events by overlining them. Notice that this notation does not allow for the representation of every possible uncertain trace: in the case of timestamp uncertainty, it can only express mutual overlapping of time intervals. However, this notation is adequate to illustrate an example for process discovery under uncertainty.

For instance, the trace $\langle \overline{a}, \{b, c\}, [d, e] \rangle$ is a trace containing 4 events, of which the first is an indeterminate event with activity label a, the second is an uncertain event that can have either b or c as activity label, and the last two events have an interval as timestamp (and the two ranges overlap). Let us consider the following event log: $\langle a, b, e, f, g, h \rangle^{80}, \langle a, \{b, c\}, [e, f], g, i \rangle^{15}, \langle a, \{b, c, d\}, [e, f], g, \overline{j} \rangle^{5}$.

For each pair of activities, we can count the minimum and maximum occurrences of a directly-follows relationship that can be observed in the log. The resulting UDFG is shown in Figure 4.

This graph can be then used to discover process models of uncertain logs via process discovery methods based on directly-follows relationships. In a previous work [8] we illustrated this principle by applying it to the Inductive Miner, a popular discovery algorithm [9]; the edges of the UDFG can be filtered via the information on the labels, in such a way that the final model can represent all possible behavior in the uncertain log, or only a part. Figure 5 shows some process models obtained through inductive mining of the UDFG, as well as a description regarding how the model relate to the original uncertain log.

UDFGs of uncertain event data are obtained on the basis of the behavior graphs of the traces in an uncertain event log, making their construction a necessary step to perform uncertain process discovery. In fact, the frequency information labeling the edges of UDFGs are obtained through a search among the possible connections within the behavior graphs of all the traces in an uncertain log.

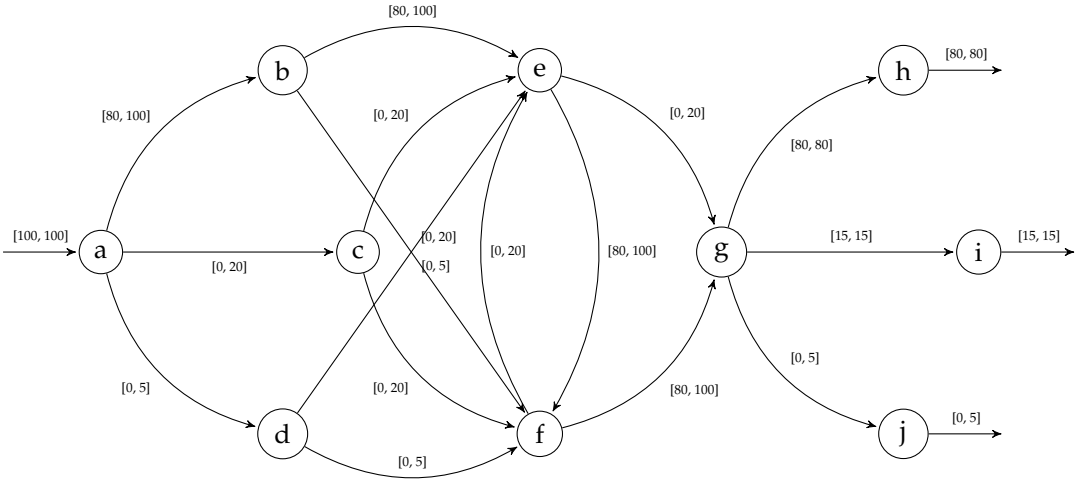
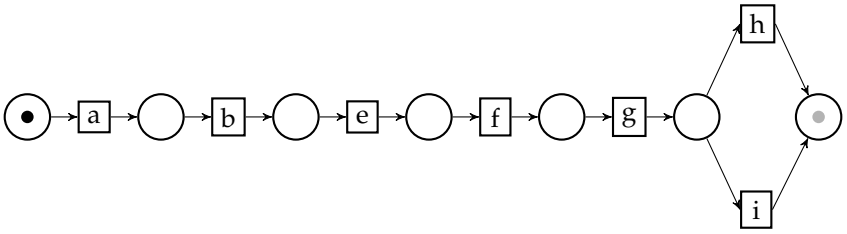
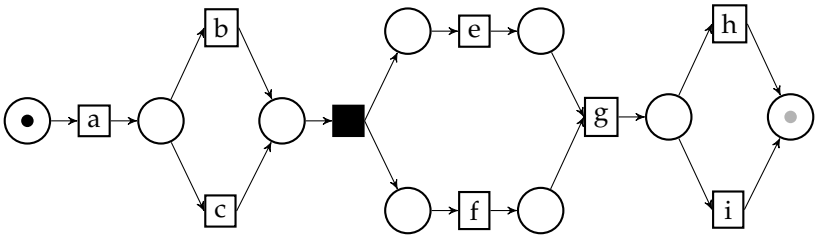


Figure 4. The *Uncertain Directly-Follows Graph* (UDFG) computed based on the uncertain event log $\langle a, b, e, f, g, h \rangle^{80}$, $\langle a, \{b, c\}, [e, f], g, i \rangle^{15}$, $\langle a, \{b, c, d\}, [e, f], g, \bar{j} \rangle^{5}$. The arcs are labeled with the minimum and maximum number of directly-follows relationship observable between activities in the corresponding trace. Notice the large amount of connections extracted from a single and rather short trace. Uncertain directly-follows relationships are inferred from the behavior graphs of the traces in the log. The construction of this object is necessary to perform automatic process discovery over uncertain event data.

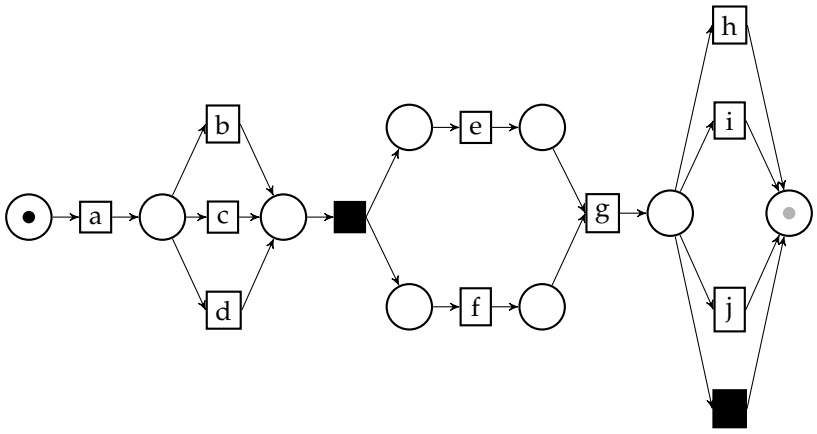


(**a**) A process model that can only replay the relationships appearing in the certain parts of the traces in the uncertain log. Here, information from uncertainty has been excluded completely.



(**b**) A process model that can replay some–but not all–the relationships appearing in the uncertain parts of the traces in the uncertain log. This process model mediates between representing only certain observation and representing all the possible behavior in the process.

Figure 5. *Cont.*



(**c**) A process model that can replay all possible configurations of certain and uncertain traces in the uncertain log. This process model has the highest possible replay fitness, but is also very likely to contain some noisy or otherwise unwanted behavior.

Figure 5. Three different process models for the uncertain event log $\langle a, b, e, f, g, h\rangle^{80}$, $\langle a, \{b, c\}, [e, f], g, i\rangle^{15}$, $\langle a, \{b, c, d\}, [e, f], g, \bar{j}\rangle^{5}$ obtained through inductive mining over an uncertain directly-follows graph. The different filtering parameters for the UDFG yield models with distinct features.

Thus, the construction of behavior graphs for uncertain traces is the basis of both process discovery and conformance checking on uncertain event data, since the behavior graph is a necessary processing step to mine information from uncertain traces. It is then important to be able to build the behavior graph of any given uncertain trace quickly and efficiently, in order to enable performant process discovery and conformance checking.

4. Materials and Methods

4.1. Preliminaries

Let us illustrate some basic concepts and notations, partially from [1]:

Definition 1 (Power set)**.** *The power set of a set A is the set of all possible subsets of A, and is denoted with $\mathcal{P}(A)$. $\mathcal{P}_{NE}(A)$ denotes the set of all the non-empty subsets of A: $\mathcal{P}_{NE}(A) = \mathcal{P}(A) \setminus \{\emptyset\}$.*

Definition 2 (Multiset)**.** *A multiset is an extension of the concept of set that keeps track of the cardinality of each element. $\mathcal{B}(A)$ is the set of all multisets over some set A. Multisets are denoted with square brackets, e.g., $b = [x, x, y]$, or with the cardinality of the elements as superscript, e.g., $b = [x^2, y]$. We denote the empty multiset with $[\,]$. The operator $(\cdot)$ retrieves the cardinality of an element of the multiset, e.g., $b(x) = 2$, $b(y) = 1$, $b(z) = 0$. Over multisets we define $x \in b \Leftrightarrow b(x) \geq 1$, and $set(b) = \{x \in b\}$. The multiset union $b = b_1 \uplus b_2$ is the multiset b such that for all x we have $b(x) = b_1(x) + b_2(x)$.*

Definition 3 (Sequence and permutation)**.** *Given a set X, a finite sequence over X of length n is a function $s \in X^* : \{1, \ldots, n\} \to X$, and is written as $s = \langle s_1, s_2, \ldots, s_n\rangle$. For any sequence s we define $|s| = n$, $s[i] = s_i$, $x \in s \Leftrightarrow x \in \{s_1, s_2, \ldots, s_n\}$ and $s \oplus s_0 = \langle s_1, s_2, \ldots, s_n, s_0\rangle$. A permutation of the set X is a sequence $x_{\mathcal{S}}$ that contains all elements of X without duplicates: $x_{\mathcal{S}} \in X$, $X \in x_{\mathcal{S}}$, and for all $1 \leq i \leq |x_{\mathcal{S}}|$ and for all $1 \leq j \leq |x_{\mathcal{S}}|$, $x_{\mathcal{S}}[i] = x_{\mathcal{S}}[j] \to i = j$. We denote with $\mathcal{S}_X$ all such permutations of set X. We overload the notation for sequences: given a sequence $s = \langle s_1, s_2, \ldots, s_n\rangle$, we will write $\mathcal{S}_s$ in place of $\mathcal{S}_{\{s_1, s_2, \ldots, s_n\}}$.*

Definition 4 (Transitive relation and correct evaluation order)**.** *Let X be a set of objects and R be a binary relation $R \subseteq X \times X$. R is transitive if and only if for all $x, x', x'' \in X$ we have that $(x, x') \in R \wedge (x', x'') \in$*

$R \rightarrow (x, x'') \in R$. *A correct evaluation order is a permutation* $s \in \mathcal{S}_X$ *of the elements of the set* X *such that for all* $1 \leq i < j \leq |s|$ *we have that* $(s[i], s[j]) \in R$.

Definition 5 (Strict partial order). *Let* S *be a set of objects. Let* $s, s' \in S$. *A strict partial order* $(\prec, S)$ *is a binary relation that have the following properties:*

- *Irreflexivity:* $s \prec s$ *is false.*
- *Transitivity:* $s \prec s'$ *and* $s' \prec s''$ *imply* $s \prec s''$.
- *Antisymmetry:* $s \prec s'$ *implies that* $s' \prec s$ *is false. It is implied by irreflexivity and transitivity [10].*

Definition 6 (Directed graph). *A directed graph* $G \in \mathcal{U}_G$ *is a tuple* (V, E) *where* V *is the set of vertices and* $E \subseteq V \times V$ *is the set of directed edges. The set* $\mathcal{U}_G$ *is the graph universe. A path in a directed graph* $G = (V, E)$ *is a sequence of vertices* p *such that for all* $1 < i < |p| - 1$ *we have that* $(p_i, p_{i+1}) \in E$. *We denote with* P_G *the set of all such possible paths over the graph* G. *Given two vertices* $v, v' \in V$, *we denote with* $p_G(v, v')$ *the set of all paths beginning in* v *and ending in* v': $p_G(v, v') = \{p \in P_G \mid p[1] = v \wedge p[|p|] = v'\}$. v *and* v' *are connected (and* v' *is reachable from* v*), denoted by* $v \overset{G}{\mapsto} v'$, *if and only if there exists a path between them in* G: $p_G(v, v') \neq \emptyset$. *Conversely,* $v \overset{G}{\not\mapsto} v' \Leftrightarrow p_G(v, v') = \emptyset$. *We drop the superscript* G *if it is clear from the context. A directed graph* G *is acyclic if there exists no path* $p \in P_G$ *satisfying* $p[1] = p[|p|]$.

Definition 7 (Topological sorting). *Let* $G = (V, E)$ *be an acyclic directed graph. A topological sorting [11]* $o_G = \langle v_1, v_2, \ldots, v_{|V|} \rangle \in \mathcal{S}_V$ *is a permutation of the vertices of* G *such that for all* $1 \leq i < j \leq |V|$ *we have that* $v_j \not\mapsto v_i$. *We denote with* $\mathcal{O}_G \subseteq \mathcal{S}_V$ *all such possible topological sortings over* G.

Definition 8 (Transitive reduction). *A transitive reduction [12]* $\rho \colon \mathcal{G} \rightarrow \mathcal{G}$ *of a graph* $G = (V, E)$ *is a graph* $\rho(G) = (V, E_r)$ *with* $E_r \subseteq E$ *where every pair of vertices connected in* $\rho(G)$ *is not connected by any other path: for all* $(v, v') \in E_r$, $p_G(v, v') = \{\langle v, v' \rangle\}$. $\rho(G)$ *is the graph with the minimal number of edges that maintain the reachability between edges of* G. *The transitive reduction of a directed acyclic graph always exists and is unique [12].*

This paper proposes an analysis technique on *uncertain event logs*. These execution logs contain information about uncertainty explicitly associated with event data. A taxonomy of different types of uncertain event logs and attribute uncertainty has been described in [2]; we will refer to the notion of *simple uncertainty*, which includes uncertainty without probabilistic information on the control-flow perspective: activities, timestamps, and indeterminate events.

Definition 9 (Universes). *Let* $\mathcal{U}_I$ *be the set of all the event identifiers. Let* $\mathcal{U}_C$ *be the set of all case ID identifiers. Let* $\mathcal{U}_A$ *be the set of all the activity identifiers. Let* $\mathcal{U}_T$ *be the totally ordered set of all the timestamp identifiers. Let* $\mathcal{U}_O = \{!, ?\}$, *where the "!" symbol denotes determinate events, and the "?" symbol denotes indeterminate events.*

Definition 10 (Simple uncertain events). $e = (e_i, A, t_{min}, t_{max}, o)$ *is a simple uncertain event, where* $e_i \in \mathcal{U}_E$ *is its event identifier,* $A \in \mathcal{P}_{NE}(\mathcal{U}_A$ *is the set of possible activity labels for* e, t_{min} *and* t_{max} *are the lower and upper bounds for the value of its timestamp, and* o *indicates if it is an indeterminate event. Let* $\mathcal{U}_E = (\mathcal{U}_I \times \mathcal{P}_{NE}(\mathcal{U}_A) \times \mathcal{U}_T \times \mathcal{U}_T \times \mathcal{U}_O)$ *be the set of all simple uncertain events. Over the uncertain event* $e = (e_i, A, t_{min}, t_{max}, o)$ *we define the projection functions* $\pi_a(e) = A$, $\pi_{t_{min}}(e) = t_{min}$, $\pi_{t_{max}}(e) = t_{max}$ *and* $\pi_o(e) = o$.

Definition 11 (Simple uncertain traces and logs). $\sigma \subseteq \mathcal{U}_E$ *is a simple uncertain trace if for any* $(e_i, A, t_{min}, t_{max}, o) \in \sigma$, $t_{min} < t_{max}$ *and all the event identifiers are unique.* $\mathcal{T}_U$ *denotes the universe of simple uncertain traces.* $L \subseteq \mathcal{T}_U$ *is a* simple uncertain log *if all the event identifiers in the log are unique.*

Definition 12 (Strict partial order over simple uncertain events)**.** *Let $e, e' \in \mathcal{E}_U^S$ be two simple uncertain events. $(\prec, \mathcal{E}_U^S)$ is an order defined on the universe of strongly uncertain events $\mathcal{E}_U^S$ as:*

$$e \prec e' \Leftrightarrow \pi_{t_{max}}(e) < \pi_{t_{min}}(e')$$

Definition 13 (Order-realizations of simple uncertain traces)**.** *Let $\sigma \in \mathcal{T}_U$ be a simple uncertain trace. An order-realization $\sigma_O = \langle e_1, e_2, \ldots, e_{|\sigma|} \rangle \in \mathcal{S}_\sigma$ is a permutation of the events in σ such that for all $1 \leq i < j \leq |\sigma|$ we have that $e_j \not\prec e_i$, i.e., σ_O is a correct evaluation order for σ over $(\prec, \mathcal{E}_U^S)$, and the (total) order in which events are sorted in σ_O is a linear extension of the strict partial order $(\prec, \mathcal{E}_U^S)$. We denote with $\mathcal{R}_O(\sigma)$ the set of all such order-realizations of the trace σ.*

A necessary step to allow for analysis of simple uncertain traces is to obtain their *behavior graph*. A behavior graph is a directed acyclic graph that synthesizes the information regarding the uncertainty on timestamps contained in the trace.

Definition 14 (Behavior graph)**.** *Let $\sigma \in \mathcal{T}_U$ be a simple uncertain trace. Let the identification function $id \colon \sigma \to \{1, 2, \ldots, |\sigma|\}$ be a bijection between the events in σ and the first $|\sigma|$ natural numbers. A behavior graph $\beta \colon \mathcal{T}_U \to \mathcal{U}_G$ is the transitive reduction of a directed graph $\rho(G)$, where $G = (V, E) \in \mathcal{U}_G$ is defined as:*

- $V = \{(id(e), \pi_a(e), \pi_o(e)) \mid e \in \sigma\}$
- $E = \{(v, w) \mid v, w \in V \wedge \pi_{t_{max}}(v) < \pi_{t_{min}}(w)\}$

The set of topological sortings of a behavior graph $\beta(\sigma)$ corresponds to the set of all the order-realizations of the trace σ:

A technical note: this definition for the nodes of the behavior graph is slightly different from the one in [2], to simplify the notation in algorithms. The two definitions are functionally identical.

Figures 6 and 7 show the transitive reduction operation on the running example.

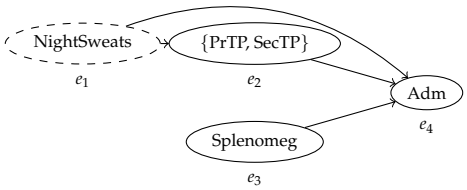


Figure 6. The behavior graph of the trace in Table 1 before applying the transitive reduction. All the nodes in the graph are pairwise connected based on precedence relationships; pairs of nodes for which the order is unknown are not connected.

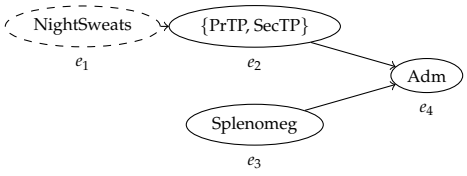


Figure 7. The same behavior graph after the transitive reduction. The arc between e_1 and e_4 is removed, since they are reachable through e_2. This graph has a minimal number of arcs while conserving the same reachability relationship between nodes.

The semantics of a behavior graph can efficaciously communicate time and order information concerning the time relationships among events in the corresponding uncertain trace in a compact manner. For a behavior graph $\beta(\sigma) = (V, E)$ and two events $e_1 \in \sigma, e_2 \in \sigma$, $(e_1, e_2) \in E$ holds if and only if e_1 is immediately followed by e_2 for some possible values of the timestamps of the events in the

trace. A consequence of this fact is that if a pair of events in the graph are unreachable, they might have occurred in any order.

Definition 14 is meaningful and clear from a theoretical point of view. It rigorously defines a behavior graph and the semantics of its parts. Although helpful to understand the function of behavior graphs, obtaining them from process traces following this definition—that is, using the transitive reduction—is inefficient and slow. This hinders the analysis of logs with a large number of events, and with longer traces. It is nonetheless possible to build behavior graphs from process traces in a faster and more efficient way.

4.2. Efficient Construction of Behavior Graphs

The set of steps to efficiently create a behavior graph from an uncertain trace is separated into two distinct phases, described by Algorithms 1 and 2. An uncertain event e is associated with a time interval which is determined by two values: minimum and maximum timestamp of that event $\pi_{t_{\min}}(e)$ and $\pi_{t_{\max}}(e)$. If an event e has a certain timestamp, we have that $\pi_{t_{\min}}(e) = \pi_{t_{\max}}(e)$.

We will examine here the effect of Algorithms 1 and 2 on a running example, the process trace shown in Table 4. Notice that in this running example, no uncertainty on activity labels nor indeterminate events are present: this is because of the fact that the topology of a behavior graph only depends on the (uncertain) timestamps in the events belonging to the corresponding trace.

Algorithm 1: TIMESTAMPLIST(σ)

Input : An uncertain trace σ.
Output : The list of timestamps $\mathcal{L}$ of σ.

$\mathcal{L}^* \leftarrow \langle\,\rangle$; `// Support list`
$\mathcal{L} \leftarrow \langle\,\rangle$; `// List of event attributes`
$\mathbb{E} \leftarrow$ SORT(σ) ; `// Sorts uncertain events by minimum timestamp`
$i \leftarrow 1$
while $i \leq |\mathbb{E}|$ **do**
  $\mathcal{L}^* \leftarrow \mathcal{L}^* \oplus (\pi_{t_{\min}}(e), i, e, \text{'MIN'})$
  $\mathcal{L}^* \leftarrow \mathcal{L}^* \oplus (\pi_{t_{\max}}(e), i, e, \text{'MAX'})$
  $i \leftarrow i + 1$
end
SORT($\mathcal{L}^*$) ; `// Sorts the list based on timestamp value`
$i \leftarrow 1$
while $i \leq |\mathcal{L}^*|$ **do**
  $(t, id, e, type) \leftarrow \mathcal{L}^*[i]$
  $\mathcal{L} \leftarrow \mathcal{L} \oplus (id, \pi_a(e), \pi_o(e), type)$
  $i \leftarrow i + 1$
end
return $\mathcal{L}$

Table 4. Running example for the creation of the behavior graph.

Case ID	Event ID	Activity	Timestamp	Event Type
872	e_1	a	5 December 2011	!
872	e_2	b	[6 December 2011, 10 December 2011]	!
872	e_3	c	7 December 2011	!
872	e_4	d	[8 December 2011, 11 December 2011]	!
872	e_5	e	9 December 2011	!
872	e_6	f	[12 December 2011, 13 December 2011]	!

Algorithm 2: BEHAVIORGRAPH(TIMESTAMPLIST(σ))

Input : The list $\mathcal{L} =$ TIMESTAMPLIST(σ) of an uncertain trace σ.
Output : The behavior graph $\beta(\sigma) = (V, E)$.

$V \leftarrow \{(id, \pi_a(e), \pi_o(e)) \mid (id, \pi_a(e), \pi_o(e), type) \in \mathcal{L}\}$
$E \leftarrow \varnothing$
$i \leftarrow 1$
while $i < |\mathcal{L}|$ **do**
  $(id, a, o, type) \leftarrow \mathcal{L}[i]$
  if $type =$ *'MAX'* **then**
    $j \leftarrow i + 1$
    while $j \leq |\mathcal{L}|$ **do**
      $(id^*, a^*, o^*, type^*) \leftarrow \mathcal{L}[j]$
      if $type^* =$ *'MIN'* **then**
        $E \leftarrow E \cup \{((id, a, o), (id^*, a^*, o^*))\}$
      else if $((id, a, o), (id^*, a^*, o^*)) \in E$ **then**
        break
      end
      $j \leftarrow j + 1$
    end
  end
  $i \leftarrow i + 1$
end
return (V, E)

The construction of the graph relies on a preprocessing step shown in Algorithm 1, where a support list $\mathcal{L}$ is created (lines 4–8). Every entry in this list is a tuple of four elements. For each event e in the trace, we insert two entries in the list—one for each timestamp $\pi_{t_{\min}}$ and $\pi_{t_{\max}}$ appearing in a trace. The four elements in each tuple contained in the list are:

- an *identifier*, which in the list construction is an integer representing the rank of the uncertain event by minimum timestamp (computed in line 3);
- the activity labels associated with the event $\pi_a(e)$;
- the attribute $\pi_o(e)$, which will carry the information regarding indeterminate events;
- the type of timestamp that generated this entry—if it is a minimum or maximum of an interval.

As we can see, the list is designed to contain all information about an uncertain event except the values of minimum and maximum timestamps, which we use to sort the list (line 9) and then discard prior to returning the list (lines 10–15).

The events of the trace in Table 4 are represented in the list $\mathcal{L}^*$ by entries shown in Table 5. These entries are then sorted by Algorithm 1 yielding the following list $\mathcal{L}$:

$$\mathcal{L} = \langle (1, \{a\}, !, \text{'MIN'}), (1, \{a\}, !, \text{'MAX'}), (2, \{b\}, !, \text{'MIN'}), (3, \{c\}, !, \text{'MIN'}),$$
$$(3, \{c\}, !, \text{'MAX'}), (4, \{d\}, !, \text{'MIN'}), (5, \{e\}, !, \text{'MIN'}), (5, \{e\}, !, \text{'MAX'}),$$
$$(2, \{b\}, !, \text{'MAX'}), (4, \{d\}, !, \text{'MAX'}), (6, \{f\}, !, \text{'MIN'}), (6, \{f\}, !, \text{'MAX'})\rangle$$

Table 5. Entries for the list $\mathcal{L}$ generated by each event in the uncertain trace. Every event e has two associated entries, one marked as 'MIN' and the other as 'MAX'. Each entry is a 4-uple containing an integer that acts as event identifier, the set of possible activity labels $\pi_a(e)$ of the uncertain event, the indeterminate event attribute $\pi_o(e)$, and the type of timestamp ('MIN' or 'MAX').

Event	List $\mathcal{L}^*$ Entry (Minimum Timestamp)	List $\mathcal{L}^*$ Entry (Maximum Timestamp)
e_1	(5 December 2011, 1, $\{a\}$, !, 'MIN')	(5 December 2011, 1, $\{a\}$, !, 'MAX')
e_2	(6 December 2011, 2, $\{b\}$, !, 'MIN')	(10 December 2011, 2, $\{b\}$, !, 'MAX')
e_3	(7 December 2011, 3, $\{c\}$, !, 'MIN')	(7 December 2011, 3, $\{c\}$, !, 'MAX')
e_4	(8 December 2011, 4, $\{d\}$, !, 'MIN')	(8 December 2011, 4, $\{d\}$, !, 'MAX')
e_5	(9 December 2011, 5, $\{e\}$, !, 'MIN')	(9 December 2011, 5, $\{e\}$, !, 'MAX')
e_6	(12 December 2011, 6, $\{f\}$, !, 'MIN')	(13 December 2011, 6, $\{f\}$, !, 'MAX')

One of the purposes the list $\mathcal{L}$ serves is gathering the structural information to create the behavior graph; in fact, visiting the list in order is equivalent of sweeping the events of the trace on the time dimension, encountering each timestamp (minimum or maximum) sorted through time. We can visualize this on the Gantt diagram representation of the trace of Table 4, visible in Figure 8.

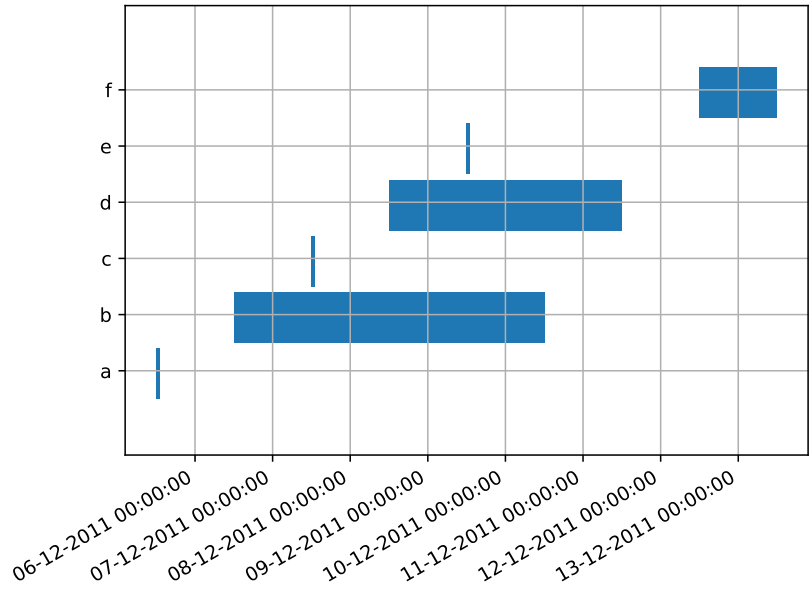


Figure 8. A Gantt diagram visualizing the time perspective of the events in Table 4. The horizontal blue bars represent the interval of possible timestamps of uncertain events: such interval is ample for the event with activity label "c", which has an uncertain timestamp, and is narrow to indicate a precise point in time for the other events. This diagram can show the order relationship between events in a trace, as well as the dimensions of their interval of possible timestamps in scale.

Every segment representing an uncertain event in the diagram is translated by TIMESTAMPLIST into two entries in a sorted list, representing the two extremes of the segment. Events without an uncertain timestamp collapse into a single point in the diagram, and their corresponding two entries in the list are characterized by the same timestamp.

Now, let us examine Algorithm 2. The idea leading the algorithm is to analyze the time relationship among uncertain events in a more precise manner, as opposed to adding a large number of edges to the graph and then removing them via transitive reduction. This is attained by searching all the viable successors of each event in the sorted timestamp list $\mathcal{L}$. We scan the list $\mathcal{L}$ with two nested loops, and we use the inner loop to look for successors of the entry selected by the outer loop. According to the semantics of behavior graphs, events with overlapping intervals as timestamps must not be connected

by a path; thus, we draw outgoing edges from an event only when, reading the list, we arrive at a point in time in which the event has certainly occurred. This is the reason outgoing edges are not drawn when inspecting minimum timestamps (line 6) and incoming edges are not drawn when inspecting maximum timestamps (line 10).

First, we initialize the set of nodes with all the triples $(id, \pi_a(e), \pi_o(e))$ in the entries of $\mathcal{L}$, and we initialize the edges with an empty set (lines 1–2). For each maximum timestamp that we encounter in the list, we start searching for successors in the following entries (lines 3–9), so we proceed in looking for the successors of $(id, a, o, type)$ only if $type =$ 'MAX'.

If, while searching for successors of the entry $(id, a, o,$ 'MAX'$)$, we encounter the entry $(id^*, a^*, o^*, type^*)$ corresponding to a minimum timestamp ($type^* =$ 'MIN'), we connect (id, a, o) and (id^*, a^*, o^*) in the graph, since their timestamps do not have any possible value in common. The search for successors must continue, since it is possible that other events took place before the maximum timestamp of the event corresponding to $(id^*, a^*, o^*, type^*)$. This configuration occurs for events e_1 and e_3 in Table 4. As can be seen in Figure 8, e_3 can indeed follow e_1, but the still undiscovered event e_2 is another possible successor for e_1.

If the entry $(id^*, a^*, o^*, type^*)$ corresponds to a maximum timestamp (line 12), so $type^* =$ 'MAX', there are two separate situations to consider. Case 1: (id, a, o) was not already connected to (id^*, a^*, o^*). Then, the timestamps of the events corresponding to (id, a, o) and (id^*, a^*, o^*) overlap with each other—if they did not, the two nodes would have already been connected, since we would have encountered $(id^*, a^*, o^*,$ 'MIN'$)$ from $(id, a, o,$ *'MAX'*$)$ before encountering $(id^*, a^*, o^*,$ 'MAX'$)$. Thus, (id, a, o) must not be connected to (id^*, a^*, o^*) and the search must continue. Events e_3 and e_4 are an example: when the maximum timestamp of e_4 is encountered during the search for the successor of e_3, the two are not connected, so the search for a viable successor of e_3 has to continue. Case 2: (id, a, o) and (id^*, a^*, o^*) are already connected. This means that we had already encountered $(id^*, a^*, o^*,$ 'MIN'$)$ during the search for the successors of (id, a, o). Since the entire time interval representing the possible timestamp of the event associated with (id^*, a^*, o^*) is detected after the occurrence of (id, a, o), there are no further events to consider as successors of (id, a, o) and the search stops (line 13). In the running example, this happens between e_5 and e_6: when searching for the successors of e_5, we first connect it with e_6 when we encounter its minimum timestamp; we then encounter its maximum timestamp, so no other successive event can be a successor for e_5. This concludes the walkthrough of the procedure, which shows why Algorithms 1 and 2 can be used to correctly compute the behavior graph of a trace. The behavior graph of the trace in Table 4 obtained through this procedure is shown in Figure 9.

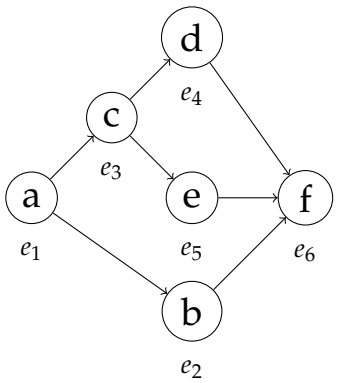


Figure 9. The behavior graph of the trace in Table 4.

Let us now prove, in more formal terms, the correctness of these algorithms. We will show that the procedures BehaviorGraph and TimestampList are able to construct a behavior graph with the semantics illustrated in Definition 14.

Theorem 1 (Correctness of the behavior graph construction). *Let $\sigma \in \mathcal{T}_U$ be an uncertain trace. Let $bg = (V, E) = \textsc{BehaviorGraph}(\textsc{TimestampList}(\sigma))$ be the behavior graph of σ obtained through Algorithms 1 and 2. The graph bg follows the behavior graph semantics: for all pairs of events $e \in \sigma$ and $e' \in \sigma$ such that $id(e) = e_{id}$, $\pi_a(e) = e_a$, $\pi_o(e) = e_o$, $id(e') = e'_{id}$, $\pi_a(e') = e'_a$, $\pi_o(e') = e'_o$, we have that the node (e_{id}, e_a, e_o) is connected to the node (e'_{id}, e'_a, e'_o) if and only if $\pi_{t_{max}}(e) < \pi_{t_{min}}(e')$ and there exists no event $e'' \in \sigma$ such that $\pi_{t_{max}}(e) < \pi_{t_{min}}(e'') \leq \pi_{t_{max}}(e'') < \pi_{t_{min}}(e')$. Thus, $bg = \beta(\sigma)$.*

Proof. Let us first define a suitable *id* function for the behavior graph using the list $\mathbb{E}$ created in $\textsc{TimestampList}(\sigma)$. For all events $e^* \in \sigma$ and for $i \in \mathbb{N}$ such that $\mathbb{E}[i] = e^*$, we define $id(e^*) = i$. Since *id* is just an enumeration of the events in σ, it is trivially bijective.

($\Leftarrow$) Assume $\pi_{t_{\max}}(e) < \pi_{t_{\min}}(e')$. By construction, we have that $\mathcal{L} = \langle \dots, (e_{id}, e_a, e_o, \text{'MAX'}), \dots, (e'_{id}, e'_a, e'_o, \text{'MIN'}), \dots \rangle$. The checks in line 6 and line 10 only allow for edges to be linked from entries of type 'MAX' to entries of type 'MIN' that only appear in a later position in the list $\mathcal{L}$. Thus, the configuration $\pi_{t_{\max}}(e) < \pi_{t_{\min}}(e')$ is a strict prerequisite for (e_{id}, e_a, e_o) and (e'_{id}, e'_a, e'_o) to be connected: $((e_{id}, e_a, e_o), (e'_{id}, e'_a, e'_o)) \in E \Rightarrow \pi_{t_{\max}}(e) < \pi_{t_{\min}}(e')$.

($\Rightarrow$) Assume $\pi_{t_{\max}}(e) < \pi_{t_{\min}}(e')$, and that the algorithm is currently searching the successors for the entry $(e_{id}, e_a, e_o, \text{'MAX'})$. Eventually, the inner loop will consider as a successor the entry $(e'_{id}, e'_a, e'_o, \text{'MIN'})$, and since it is of type 'MIN', (e_{id}, e_a, e_o) and (e'_{id}, e'_a, e'_o) will necessarily be connected unless the algorithm executes the `break` at line 13. To execute it, the algorithm needs to find a list entry $(e''_{id}, e''_a, e''_o, \text{'MAX'})$ such that there already exist an arc between (e_{id}, e_a, e_o) and (e''_{id}, e''_a, e''_o), and this is only possible if $(e''_{id}, e''_a, e''_o, \text{'MIN'})$ has been encountered while searching for successors of (e_{id}, e_a, e_o). This implies that

$$\mathcal{L} = \langle \dots, (e_{id}, e_a, e_o, \text{'MAX'}), \dots, (e''_{id}, e''_a, e''_o, \text{'MIN'}), \dots$$
$$\dots, (e''_{id}, e''_a, e''_o, \text{'MAX'}), \dots, (e'_{id}, e'_a, e'_o, \text{'MIN'}), \dots \rangle$$

which by construction of $\mathcal{L}$, is only possible if there exist some $e'' \in \sigma$ such that

$$\pi_{t_{\max}}(e) < \pi_{t_{\min}}(e'') \leq \pi_{t_{\max}}(e'') < \pi_{t_{\min}}(e')$$

□

As mentioned earlier, the procedure of constructing a behavior graph has been structured in two different algorithms specifically to enable further optimization in processing uncertain process trace. This becomes evident once we consider the problem of converting in behavior graphs all the traces in an event log, as opposed as one single uncertain trace.

First, it is important to notice that different uncertain traces can have the same list $\mathcal{L}$. Similarly to directly-follows relationships in more classical process mining, which can ignore the amount of time in absolute terms elapsed between two consecutive events, specific values of timestamps in an uncertain trace are not necessarily meaningful with respect to the connection in the behavior graph; their order, conversely, is crucial.

This fact enables further optimization at the log level. The construction of the list $\mathcal{L}$ in $\textsc{TimestampList}(\sigma)$ is engineered in a way that allows for computing the behavior graph without direct lookup to the events in the trace. This implies that it is possible to extract a multiset of lists $\mathcal{L}$ from the event log, and to compute the conversion to behavior graph only for the set of lists induced by this multiset. This allows the saving of computation time in converting an entire event log to behavior graphs; furthermore, it enables a more compact representation of the log in memory, since we only need to store a smaller number of graphs to represent the whole log.

The procedure to efficiently convert an event log into graphs is detailed in Algorithm 3.

These considerations allow us to extend to the uncertain scenario some concepts that are essential in classical process mining. First, we can now derive the definition of *variant*, highly important for preexisting process mining techniques, to uncertain event data.

Algorithm 3: PROCESSUNCERTAINLOG

Input : An uncertain log L.
Output : A multiset of behavior graphs BG.
$ML \leftarrow [\,]$
$\mathcal{V}_L \leftarrow [\,]$
for $\sigma \in L$ **do**
  $ML \leftarrow ML \uplus [\text{TIMESTAMPLIST}(\sigma)]$
end
for $\mathcal{L} \in ML$ **do**
  $\mathcal{V}_L \leftarrow \mathcal{V}_L \uplus [\text{BEHAVIORGRAPH}(\mathcal{L})^{ML(\mathcal{L})}]$
end
return BG

Definition 15 (Uncertain variants)**.** *Let $L \subseteq \mathcal{T}_U$ be a simple uncertain event log. The variants of L denoted by $\mathcal{V}_L$, are the multisets of behavior graphs for the uncertain traces in L, and are computed with* PROCESSUNCERTAINLOG(L).

The computational advantage in representing a log through a multiset of behavior graphs is evident in the procedure described in Algorithm 2. We see that all data necessary to the creation of a behavior graph is contained in the list $\mathcal{L}$, fact that justifies the log representation method illustrated in Algorithm 3.

Lemma 1. *Two uncertain traces $\sigma_1 \in L$ and $\sigma_2 \in L$ belong to the same variant, and share the same behavior graph, if and only if they result in the same timestamp list $\mathcal{L}$:* TIMESTAMPLIST(σ_1) = TIMESTAMPLIST(σ_2).

Another central concept in process mining is the so-called *control-flow perspective* of event data. In certain process traces, where timestamps have a total order, events have a single activity label and no event is indeterminate, the control-flow information is represented by a sequence of activity labels sorted by timestamp. Although there are many analysis approaches that also account for other perspectives (e.g., the performance perspective that considers the duration of events and their distance in time, or the resource perspective that accounts for the agents that execute the activities), a vast amount of process mining techniques, including most popular algorithms for process discovery and conformance checking, rely only on the control-flow perspective of a process. Analogously, behavior graphs carry over the control-flow information of an uncertain trace: instead of describing the flow of events like their certain counterpart, the behavior graph describes all possible flows of events in the uncertain trace.

5. Asymptotic Complexity

In this section, we will provide some values for the asymptotic complexity of the algorithms seen in this paper.

In a previous paper [2] we introduced the concept of behavior graph for the representation of uncertain event data, together with a method to obtain such graphs. Definition 14 describes such a baseline method for the creation of the behavior graph consisting of two main parts: the construction of the starting graph and the computation of its transitive reduction. Let us consider an uncertain process trace $\sigma \in \mathcal{T}_U$ with $|\sigma| = n$ events, and the graph $G = (V, E)$ generated in Definition 14 before the transitive reduction.

The starting graph is created by inspecting the time relationship between every pair of events; this corresponds to checking if an edge exists between each pair of vertices in G, which needs $\mathcal{O}(n^2)$ time.

The transitive reduction of graphs can be obtained through many methods. A simple and efficient method to compute the transitive reduction on sparse graphs is to test reachability through a search

(either breadth-first or depth-first) from each edge. This method costs $\mathcal{O}(V \cdot E)$ time (here, for simplicity, we resort to a widely adopted abuse of notation in asymptotic complexity analysis: we indicate a set instead of its cardinality, e.g., we use $\mathcal{O}(V)$ in place of $\mathcal{O}(|V|)$). However, in the initial graph each event $e \in V$ has an inbound arc from each event certainly preceding e and an outbound arc to each event certainly following e. Fewer events with overlapping intervals as timestamps of uncertain events imply fewer arcs in G; the initial graph G of a trace with no uncertainty has $|E| = \frac{n(n-1)}{2} = \mathcal{O}(V^2)$ edges. Thus, except for rare, very uncertain cases, the graph G is dense.

Aho et al. [12] presented a technique to compute the transitive reduction in $\mathcal{O}(n^3)$ time, more appropriate in the case of dense graphs, and proved that the transitive reduction has the same computational complexity of the matrix multiplication problem. The problem of matrix multiplication was generally regarded as having an optimal time complexity of $\mathcal{O}(n^3)$, until Volker Strassen presented an algorithm [13] able to multiply matrices in $\mathcal{O}(n^{2.807355})$ time. Subsequent improvements have followed, by Coppersmith and Winograd [14], Stothers [15] and Williams [16]. The asymptotically fastest algorithm known to date has been illustrated by Le Gall [17] and has an execution time of $\mathcal{O}(n^{2.3728639})$. However, these faster algorithms are very seldomly used in practice, due to the existence of large constant factors in their computation time that are hidden by the asymptotic notation. Moreover, they have vast memory requirements. The Strassen algorithm is helpful in real-life applications only when applied on very large matrices [18], and the Coppersmith-Winograd algorithm and subsequent improvements are more efficient only with inputs so large that they are effectively classified as galactic algorithms [19].

Bearing in mind these considerations, for the vast majority of event logs, the most efficient way to implement the creation of the behavior graph via transitive reduction runs in $\mathcal{O}(n^2) + \mathcal{O}(n^3) = \mathcal{O}(n^3)$ time in the worst-case scenario.

It is straightforward to find upper bounds for the complexity of Algorithms 1 and 2.

Line 3 of TIMESTAMPLIST require $\mathcal{O}(n \log n)$ to be executed. Lines 5–8 require $\mathcal{O}(n)$ time. Line 9 requires $\mathcal{O}(2n \log(2n)) = \mathcal{O}(n \log n)$ time to be run. Lines 11–14 require $2n = \mathcal{O}(n)$ time to be run. Lines 1–4 and 10 have a constant cost $\mathcal{O}(1)$. Thus, TIMESTAMPLIST has a total asymptotic cost of $\mathcal{O}(1) + 2 \cdot \mathcal{O}(n \log n) + 2 \cdot \mathcal{O}(n) = \mathcal{O}(n \log n)$ in the worst-case scenario.

Let us now examine BEHAVIORGRAPH. Lines 1–3 and line 11 run in $\mathcal{O}(1)$ time. Lines 11–30 consist of two nested loops over the list $\mathcal{L}$, and we have $|\mathcal{L}| = 2n$, resulting in an asymptotic cost of $\mathcal{O}((2n)^2) = \mathcal{O}(n^2)$. The total running time for the novel construction method is then $\mathcal{O}(1) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$ time in the worst-case scenario.

We can also obtain a lower bound for the complexity in the worst-case scenario by analyzing the possible size of the output. The complete directed bipartite graph with n vertices, usually indicated with $K_{\frac{n}{2},\frac{n}{2}}$, is a DAG that has $(\frac{n}{4})^2 = \mathcal{O}(n^2)$ edges. It is easy to see that the complete bipartite graph fulfills the requirements to be a behavior graph: it is in fact acyclic, and no edge can be removed without changing the reachability of the graph—specifically, it is equivalent to its transitive reduction. We can show that a behavior graph with such a shape exists employing a simple construction: a trace composed by n events with timestamps such that the first $\frac{n}{2}$ events all have overlapping timestamps, the last $\frac{n}{2}$ also all have overlapping timestamps, and the maximum timestamp of each of the first $\frac{n}{2}$ is smaller than the minimum timestamp of each of the last $\frac{n}{2}$ events. The construction, together with an example, is illustrated in Figure 10. Since lines 11–30 of the algorithm build this graph with $\mathcal{O}(n^2)$ edges, the algorithm runs in $\Omega(n^2)$ time, and thus also in $\Theta(n^2)$ time. This also proves the asymptotic optimality of the algorithm: no algorithm to build behavior graphs can run in less than $\Theta(n^2)$ time in the worst-case scenario.

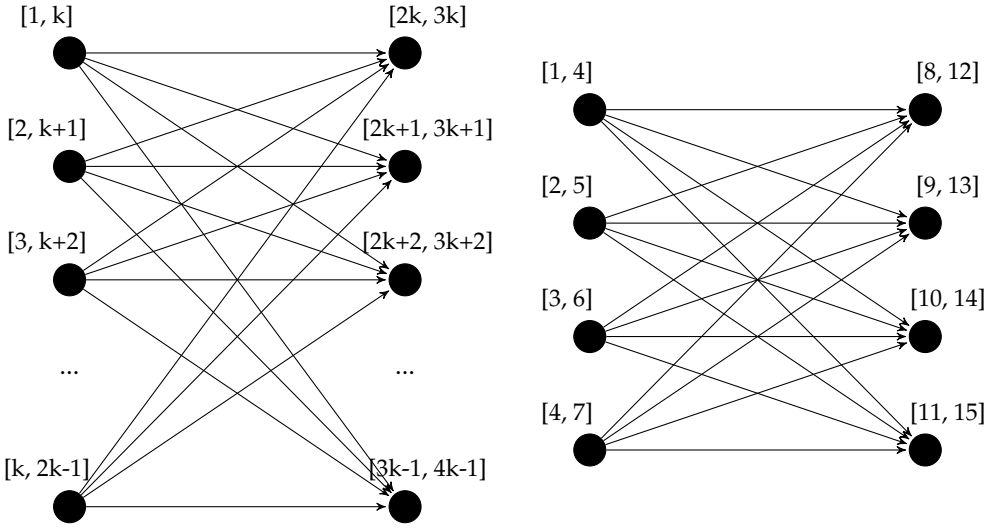


Figure 10. Construction of the class of behavior graphs isomorphic to a complete bipartite graph and an instantiated example. For any $n = 2k$, it is possible to have a behavior graph isomorphic to the graph $K_{k,k}$, which thus has a number of edges quadratic in the number of vertices.

6. Experimental Results

The formal definition of our novel construction method for the behavior graph was used to show its asymptotic speedup with respect to the construction using the transitive reduction. To empirically confirm this improvement, we built a set of experiments to measure the gain in speed and memory usage.

6.1. Performance of Behavior Graph Construction

In this section, we will show a comparison between the running time of the naïve behavior graph construction—which employs the transitive reduction—versus the improved method detailed throughout the paper. The experiments are set to investigate the difference in performance between the two algorithms, and most importantly how this difference scales when the size of the event log increases, as well as the amount of events in the log that have uncertain timestamps. In designing the experiments, we took into consideration the following research questions:

- *Q1*: how does the computation time of the two methods compare when run on logs with an increasing number of traces?
- *Q2*: how does the computation time of the two methods compare when run on logs with increasing trace lengths?
- *Q3*: how does the computation time of the two methods compare when run on logs with increasing percentages of events with uncertain timestamps?
- *Q4*: what degree of reduction in memory consumption for the representation of an uncertain log can we attain with the novel method?
- *Q5*: do the answers obtained for *Q3* hold when simulating uncertainty on real-life event data?

Both the baseline algorithm based on transitive reduction [2] and the new algorithm for the construction of the behavior graph are implemented in Python, within the PROVED project. The implementation of both methods is available online, as well as the full code for the experiments presented here (see the reference in Section 1).

For each series of experiments exploring *Q1* through *Q4*, we generate a synthetic event log with a number n of traces of length l (in number of events belonging to the trace). Uncertainty on timestamps is then artificially added to the events in the log. A specific percentage p of the events in the event log will have an uncertain timestamp, causing it to overlap with an adjacent event.

Finally, behavior graphs are built from all the traces in the event log with either algorithm, while the execution time is measured. All results in this section are presented as the mean of the measurements for 10 runs of the corresponding experiment. In the diagrams, we will label with "TrRed" the naïve method using the transitive reduction, and with "Improved" the faster algorithm illustrated in this paper. Additionally, the data series for the novel method are labeled with the relative variation in running time for each specific data point in the experiment, expressed in percentage.

To answer *Q1*, the first experiment inspects how the efficiency of the two algorithms scales with log dimension in number of traces. We generate logs with a fixed uncertainty percentage of $p = 0.5$, and trace length of $l = 20$. The number of traces in the uncertain log progressively scales from $n = 1000$ to $n = 10{,}000$. As shown in Figure 11, our proposed algorithm outperforms the baseline algorithm, showing a much smaller slope in computation time. As anticipated by the theoretical analysis, the computing time to build behavior graphs increases linearly with the number of traces in the event log for both methods; in the novel method, the constant factors are much smaller, thus producing the speedup that we can observe in the graph. Please note that in this experiment the novel method requires between 18% and 26% of the time with respect to the baseline method.

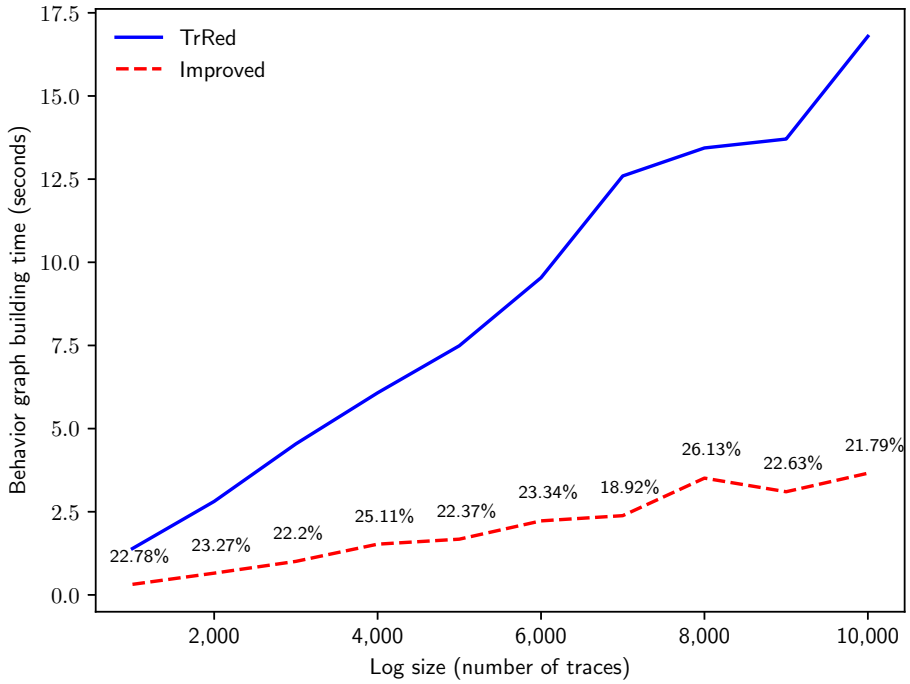


Figure 11. Time in seconds for the creation of the behavior graphs for synthetic logs with traces of length $l = 20$ events and $p = 0.5$ of uncertain events, with increasing number of traces n. The solid blue line indicates the time needed for the naïve construction; the dashed red line shows the building time of the improved algorithm, and is labeled with the relative time variation (in percentage).

The second experiment is designed to answer *Q2*. We analyze the effect of the trace length on the total time needed for behavior graph creation. Therefore, we created logs with $n = 100$ traces of increasing lengths in number of events, and added uncertain timestamps to events with $p = 0.5$. The results, illustrated in Figure 12, meet our expectations: the computation time of the baseline method scales much worse than the computation time required by our new technique, due to its cubic asymptotic time complexity. This confirms the results of the analysis of the asymptotic time complexity analysis detailed in Section 5. We can notice an order-of-magnitude increase in speed. At trace length $l = 600$, the new algorithm computes the graphs in only 0.35% of the time required by the baseline algorithm.

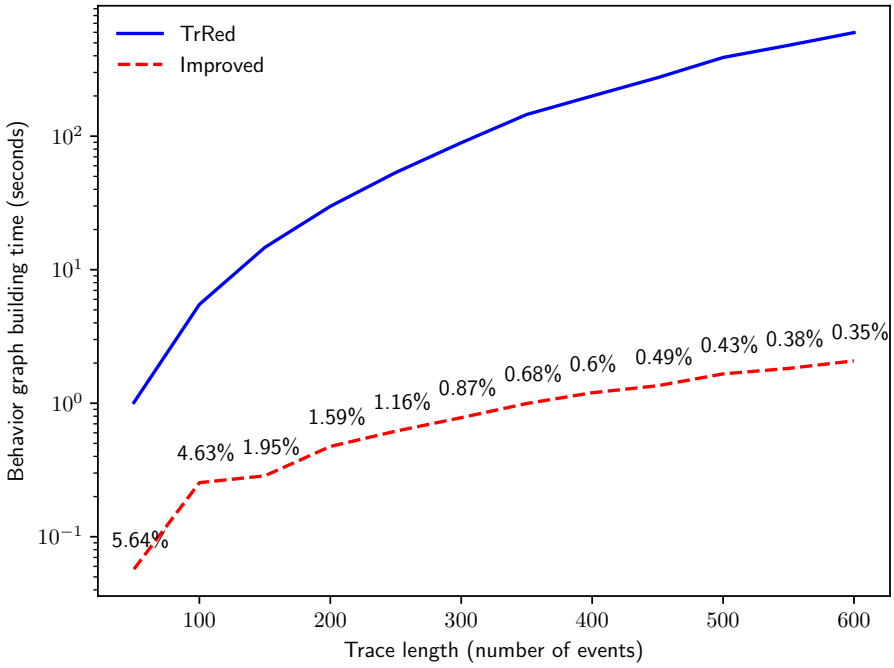


Figure 12. Time in seconds for the creation of the behavior graphs for synthetic logs with $n = 100$ traces and $p = 0.5$ of uncertain events, with increasing trace length l.

The next experiment tackles *Q3*, by inspecting the difference in execution time for the two algorithms in function of the percentage of events with an uncertain timestamp in the event log. Keeping constant the values $n = 100$ and $l = 100$, we progressively increased the percentage p of events with an uncertain timestamp and measured computation time. As presented in Figure 13, the time required for behavior graph construction remains almost constant for our proposed algorithm, while it is very slightly decreasing for the baseline algorithm. This behavior is expected, and is justified by the fact that the worst-case scenario for the baseline algorithm is a trace that has no uncertainty on the timestamp: in that case, the behavior graph is simply a chain of nodes representing the total order in a sequence of events with certain timestamps, thus the transitive reduction needs to find and remove a higher number of edges from the directed graph. This worst-case scenario occurs at $p = 0$, explaining why the computation time needed by the transitive reduction is at its highest. It is important to note, however, that for all values of p our new algorithm runs is significantly more efficient than the baseline algorithm: with $p = 0$, the new algorithm takes 0.47% of the time needed by the naïve construction, while for $p = 1$ this figure grows to 4.39%.

An additional experiment is illustrated to provide an answer to *Q4*. As with the first experiment, we increase the number of traces n in the uncertain log, while keeping the other parameters fixed: $l = 10$ and $p = 0.5$. We then perform the behavior graph construction with both methods, and we measure the memory consumption derived from the transitive reduction method (keeping in memory one behavior graph for each uncertain trace) versus the improved method (which generates a multiset of behavior graphs, one for each variant in the uncertain log).

The results are summarized in Figure 14. Please note that when n increases, more and more uncertain traces are characterized by the same behavior graph, and can then be grouped in the same variant. This allows the improved algorithm to store the uncertain log more effectively. At $n = 15{,}000$, the space needed by the multiset of behavior graphs is 59.2%, a sizable improvement in memory requirements when analyzing uncertain event logs of substantial dimensions. This improvement in memory consumption is a consequence of the new technique used in this paper to obtain the timestamp list, which enables such refinement with respect to the technique illustrated in [5].

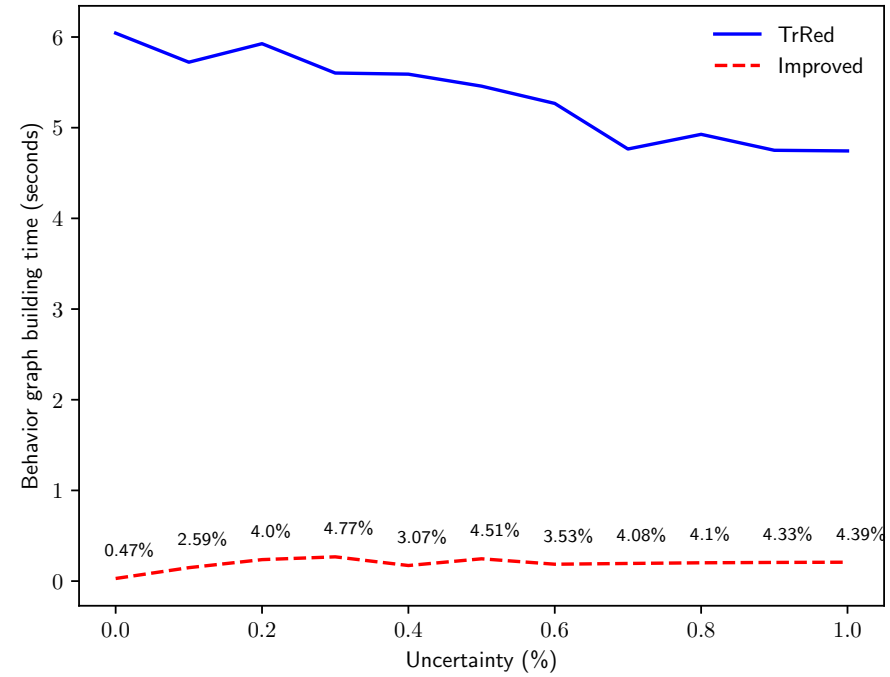


Figure 13. Time in seconds for the creation of the behavior graphs for synthetic logs with $n = 100$ traces of length $l = 100$ events, with increasing percentages of timestamp uncertainty p.

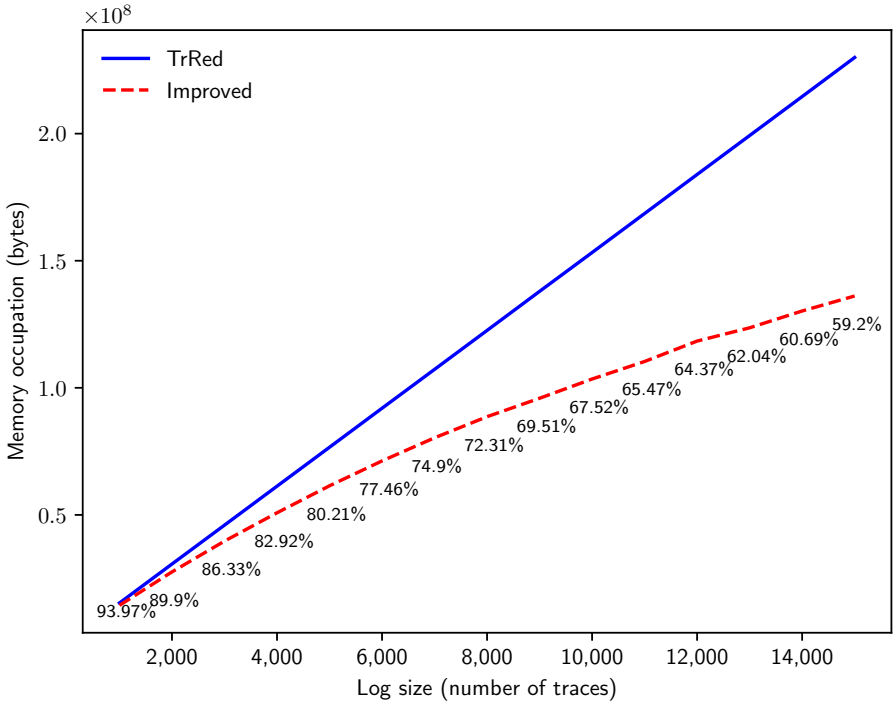


Figure 14. Memory consumption in bytes needed to store the behavior graphs for synthetic uncertain event logs with traces of length $l = 10$ events and timestamp uncertainty of $p = 0.5$, with an increasing number of traces n.

Finally, to elucidate research question *Q5* we compared the computation time for behavior graphs creation on real-life event logs, where we artificially inserted timestamp uncertainty in progressively higher percentage of uncertain events p as described for the experiments above. We considered three event logs: an event log tracking the activities of the help desk process of an Italian software company,

a log related to the management of road traffic fines in an Italian municipality, and a log from the BPI Challenge 2012 related to a loan application process.

The results, presented in Figure 15, closely adhere to the findings of the experiments with synthetically generated uncertain event data: the novel method provides a substantial speedup that remains rather stable with respect to the percentage p of uncertain events added in the log.

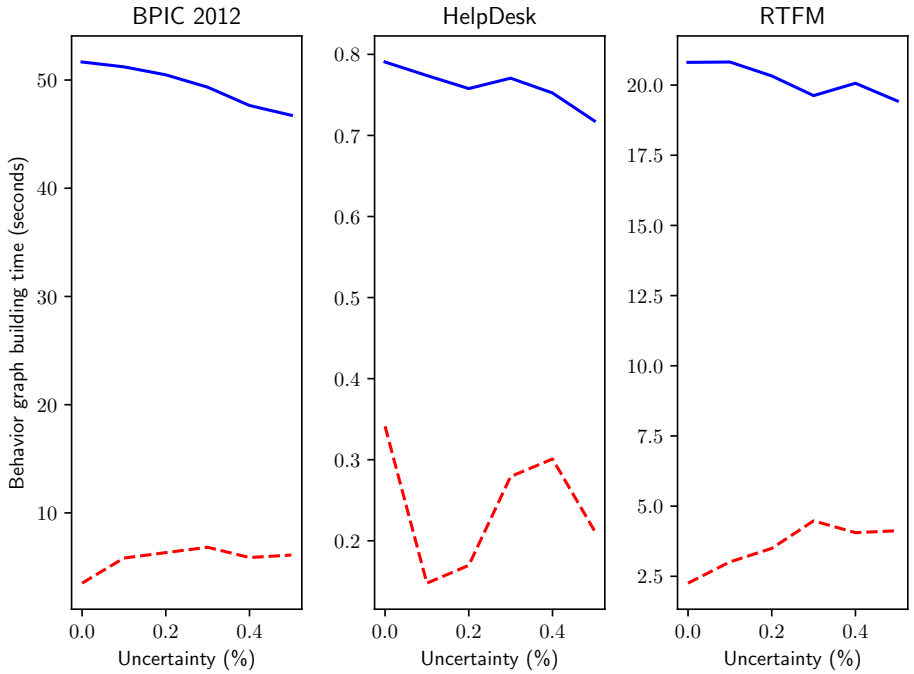


Figure 15. Execution times in seconds for real-life event logs with increasing percentages p of timestamp uncertainty.

6.2. Applications of the Behavior Graph Construction

In Section 1 we saw how building the behavior graph is a fundamental preprocessing step for both process discovery and conformance checking when dealing with uncertain event logs. In the previous section, we showed in practice how the novel algorithm presented in this paper impacts the computation time for the construction of behavior graphs. Now, let us have a glance into the effect of the speedup when applied to process mining techniques.

In this additional experiment we consider the conformance checking problem. In [2] we proposed an approach to compute upper and lower bounds for the conformance score of a trace against a reference Petri net through the alignment technique, which yields alignments for the best- and worst-case scenarios of an uncertain trace as illustrated in Section 1. The experiment is set up to assess the effect of the new behavior graph construction on the overall performance of conformance checking over uncertain data. We first generate a Petri net with t transitions, simulate a log by playing out $n = 500$ traces, and add timestamp uncertainty with $p = 0.1$. We then compute the lower bound for conformance between the uncertain event log and the Petri net used as a source, and compare the overall execution time for conformance using the two different methods for the creation of the behavior graph. In this specific experiment, we also considered the other types of uncertainty in process mining illustrated in the taxonomy of [2], as well as all types of uncertainty simulate on the same log.

The results are shown in Figure 16. We can see that on very small nets ($t = 5$), the alignment algorithm takes a short time to execute, so the speedup provided by the improved behavior graph construction has a larger impact on the total computation time (taking as little as 30.71% of the time to calculate alignments). With the increase of t, the computation time for conformance checking using the fast construction of the behavior graph appears to stabilize around 65% of the time needed if we employ

the naïve construction when considering only one type of uncertainty in isolation. This translates in a reduction of roughly 35% of computation time for the very common problem of calculating the conformance score between event data and a reference model, a significant impact on performances of concrete applications of process mining over uncertain data. When compounding all types of uncertainty we see a similar effect, although for $t = 5$ the improved method takes 52.22% of the time required by the baseline construction, a less dramatic effect than the other uncertainty settings. This is because even at such small scales, the high number of realizations of traces slow down the alignment phase in the computation.

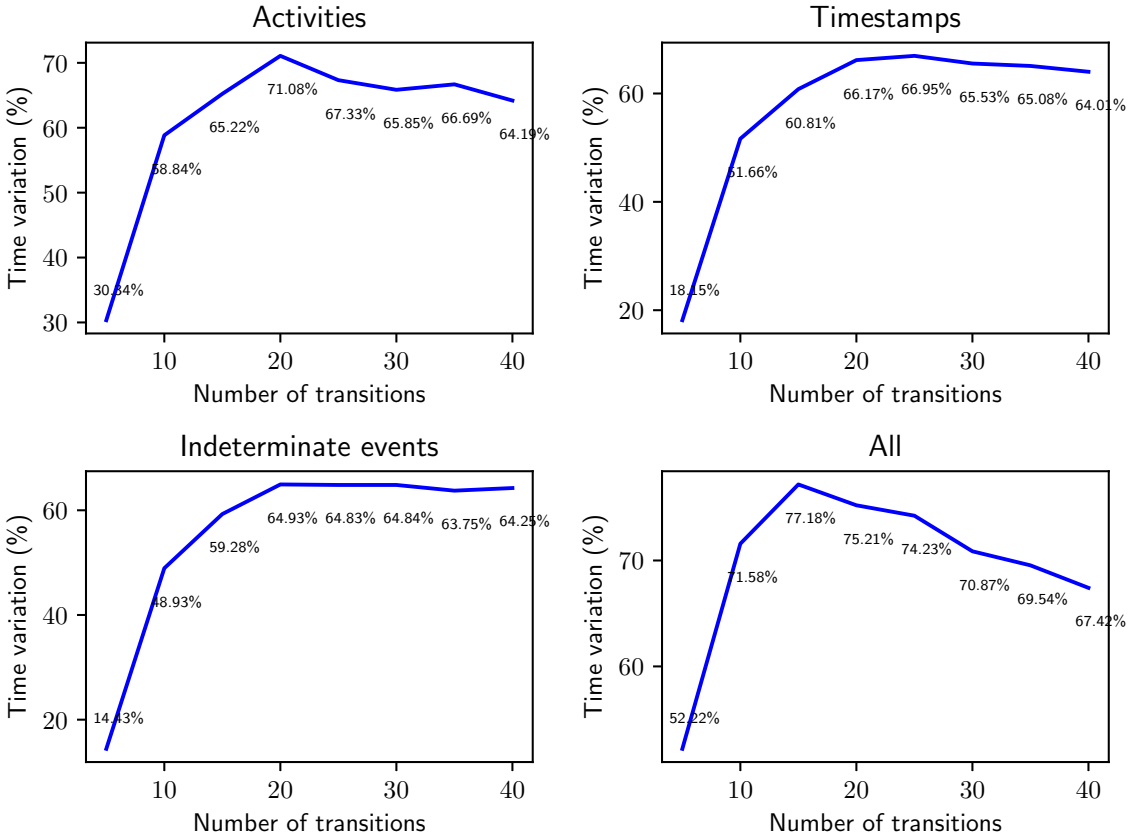


Figure 16. Relative variation in computation time obtained through the improved behavior graph construction when applied to the computation of conformance bounds between a synthetic uncertain log and a Petri net with an increasing number of transitions. The synthetic uncertain logs have $n = 500$ traces and timestamp uncertainty has been introduced with $p = 0.1$.

In evaluating this result, it is important to consider that alignments are a notoriously time-intensive technique [20], since the technique is based on an A* search on a state space that consists of pairs of the activities in the trace combined with the possible actions in the model. As a consequence, the impact of the algorithm presented in this paper is limited by the characteristics of the implementation of such alignment technique; combining it with more refined alignment algorithms would further improve the gain in speed.

In summary, the outcomes of the experiments show how our new algorithm hereby presented outperforms the previous method for creating the behavior graph on all the parameters in which the problem instance can scale in dimensions, in both the time and space dimensions. The experiment designed to answer *Q3* shows that like the naïve algorithm, our novel method being is essentially insensitive to the percentage of events with uncertain timestamps contained in a trace. This fact is also verified by the experiment associated with *Q5* on real-life data with added time uncertainty. While for

every combination of parameters we benchmarked the novel algorithm runs in a fraction of time required by the baseline method, the experiments also confirm the improvements in asymptotic time complexity demonstrated through theoretical complexity analysis.

7. Related Work

The topic of process mining analysis over uncertain event data is relatively new, and little research has been carried out. The work introducing the concept of uncertainty in process mining, together with a taxonomy of the various types of uncertainty, specifically illustrated that if a trace displays uncertain attributes, it contains behavior, which can be effectively represented through graphical models—specifically, behavior graphs and behavior nets [2]. Differently to classic process mining, where we have a clearly defined separation between data and model and between the static behavior of data and the dynamic behavior of models, the distinction between data and models becomes more unclear in the presence of uncertainty, because of the variety in behavior that affects the data. Representing traces through process models is used in [2] for the computation of upper and lower bounds for conformance scores of uncertain process traces against classic reference models. Another practical application of behavior graphs in the field of process mining over uncertain event data is presented in [8]. Behavior graphs of uncertain traces are employed to determine the number of possible directly-follows relationships between uncertain events, with the end goal of automatically discovering process models from uncertain event data.

Albeit, as said, the application of the concept of uncertainty in data to process mining is recent, the same idea has precedents in the older field of data mining. Aggarwal and Philip [21] offer an overview of the topic of uncertain data and its analysis, with a strong focus on querying. Such data is modeled based on probabilistic databases [22], a foundational concept in the setting of uncertain data mining. A branch of data mining particularly close to process mining is frequent itemsets mining: an efficient algorithm to search for frequent itemsets over uncertain data, the U-Apriori, have been described by Chui et al. [23].

Behavior graphs are Directed Acyclic Graphs (DAGs), which are widely used throughout many areas of science to represent with a graph-like model dependencies, precedence relationships, time information, or partial orders. They are effectively used in circular dependency analysis in software [24], probabilistic graphical models [25], dynamic graphs analytics [26], and compiler design [27]. In process mining, *Conditional Partial Order Graphs* (CPOGs)—which consist of collections of DAGs—have been exploited by Mokhov et al. [28] to aid the task of process discovery.

We have seen throughout the paper that uncertainty on the timestamp dimension—specifically, representing at which time an event occurred with an interval of possible timestamps—generates, on the precedence relationships of events, a partial order. Although uncertainty research in process mining provides a novel justification of partial ordering that spawns from specific attribute values, the idea of having a partial order instead of a total order among events in a trace has precedents in process mining research. Lu et al. [29,30] examined the problem of conformance checking through alignments in the case of partially ordered traces, and developed a construct to represent conformance called a *p-alignment*. Genga et al. [31] devised a method to identify highly frequent anomalous patterns in partially ordered process traces. More recently, van der Aa et al. [32] developed a probabilistic infrastructure that allows an inference of the most likely linear extension of a partial order between events in a trace, with the goal of "resolving" the partial order.

An important aspect to notice is that conformance checking over uncertain event data is not to be confused with stochastic conformance checking, which concerns measuring conformance of certain event data against models enriched with probabilistic information. The probabilities decorating a stochastic model do not derive from uncertainties in event data, but rather from frequency of activities [33] or from performance indicators [34].

A review of related work on the topic of the asymptotic complexity of the transitive reduction and the equivalent problem of matrix multiplication is provided with the complexity analysis of the algorithms examined by this paper, in Section 5.

8. Conclusions

The creation of the behavior graphs—a graphical structure of paramount importance for the analysis of uncertain data in the domain of process mining—plays a key role as initial processing step for both conformance checking and process discovery of process traces containing events with timestamp uncertainty, the most critical type of uncertain behavior. It allows, in fact, to represent the time relationship between uncertain events, which can be in a partial order. The behavior graph also carries the information regarding other types of uncertainty, like uncertain activity labels and indeterminate events. Such a representation is vital to establish which possible sequence of events in an uncertain trace most adhere to the behavior prescribed by a reference model, thereby enabling conformance checking; and to measure the number of possible occurrences of the directly-follows relationship between activities in an event log, making process discovery over uncertainty possible. Extracting behavior graphs from uncertain event data is thus concomitantly crucial and time consuming. In this paper, we show an improvement for the performance of uncertainty analysis by proposing a new algorithm that enables the creation of behavior graphs in quadratic time in the number of events in the trace. This novel method additionally allows for the representation of an uncertain log as a multiset of behavior graphs, which relevance is two-fold: it allows the representation of the control-flow information of an uncertain event log in a more compact manner by using less memory, and naturally extends the concept of variant—central throughout the discipline of process mining—to the uncertain domain. We proved the correctness of this novel algorithm, we showed asymptotic upper and lower bounds for its time complexity, and implemented performance experiments for this algorithm that effectively show the gain in computing speed it entails in real-world scenarios.

Author Contributions: The conceptualization and methodology of this research was a collaborative effort shared among all authors. Other tasks were divided as follows: software and experiment implementation, result analysis and draft preparation, M.P.; editing, critical reviewing, supervising, M.S.U. and W.M.P.v.d.A.; funding acquisition, W.M.P.v.d.A. All authors have read and agreed to the published version of the manuscript.

Funding: We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research interactions.

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

References

1. Van der Aalst, W.M.P. *Process Mining: Data Science in Action*; Springer: Berlin/Heidelberg, Germany, 2016.
2. Pegoraro, M.; van der Aalst, W.M.P. Mining uncertain event data in process mining. In Proceedings of the 2019 International Conference on Process Mining (ICPM), Aachen, Germany, 24–26 June 2019; pp. 89–96.
3. Van Der Aalst, W.; Adriansyah, A.; De Medeiros, A.K.A.; Arcieri, F.; Baier, T.; Blickle, T.; Bose, J.C.; Van Den Brand, P.; Brandtjen, R.; Buijs, J.; et al. Process mining manifesto. In *International Conference on Business Process Management*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 169–194.
4. Kurniati, A.P.; Rojas, E.; Hogg, D.; Hall, G.; Johnson, O.A. The assessment of data quality issues for process mining in healthcare using Medical Information Mart for Intensive Care III, a freely available e-health record database. *Health Inform. J.* **2019**, *25*, 1878–1893. [CrossRef] [PubMed]
5. Pegoraro, M.; Uysal, M.S.; van der Aalst, W.M.P. Efficient construction of behavior graphs for uncertain event data. In *International Conference on Business Information Systems (BIS)*; Springer: Berlin/Heidelberg, Germany, 2020.
6. Berti, A.; van Zelst, S.J.; van der Aalst, W.M.P. Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science. *arXiv* **2019**, arXiv:1905.06169.
7. Adriansyah, A.; van Dongen, B.F.; van der Aalst, W.M.P. Towards robust conformance checking. In *International Conference on Business Process Management*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 122–133.

8. Pegoraro, M.; Uysal, M.S.; van der Aalst, W.M.P. Discovering Process Models from Uncertain Event Data. In *International Conference on Business Process Management*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 238–249.
9. Leemans, S.J.J.; Fahland, D.; van der Aalst, W.M.P. Discovering block-structured process models from event logs-a constructive approach. In *International Conference on Applications and Theory of Petri Nets and Concurrency*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 311–329.
10. Flaška, V.; Ježek, J.; Kepka, T.; Kortelainen, J. Transitive closures of binary relations. I. *Acta Univ. Carol. Math. Phys.* **2007**, *48*, 55–69.
11. Kalvin, A.D.; Varol, Y.L. On the generation of all topological sortings. *J. Algorithms* **1983**, *4*, 150–162. [CrossRef]
12. Aho, A.V.; Garey, M.R.; Ullman, J.D. The transitive reduction of a directed graph. *SIAM J. Comput.* **1972**, *1*, 131–137. [CrossRef]
13. Strassen, V. Gaussian elimination is not optimal. *Numer. Math.* **1969**, *13*, 354–356. [CrossRef]
14. Coppersmith, D.; Winograd, S. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.* **1990**, *9*, 251–280. [CrossRef]
15. Stothers, A.J. On the Complexity of Matrix Multiplication. Ph.D. Thesis, University of Edinburgh, Edinburgh, UK, 2010.
16. Williams, V.V. Multiplying matrices faster than Coppersmith-Winograd. In Proceedings of the ACM Symposium on Theory of Computing (STOC), New York, NY, USA, 19–22 May 2012; Volume 12, pp. 887–898.
17. Le Gall, F. Powers of tensors and fast matrix multiplication. In Proceedings of the 39th international symposium on symbolic and algebraic computation, Kobe, Japan, 23–25 July 2014; pp. 296–303.
18. D'Alberto, P.; Nicolau, A. Using recursion to boost ATLAS's performance. In *High-Performance Computing*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 142–151.
19. Le Gall, F. Faster algorithms for rectangular matrix multiplication. In Proceedings of the 53rd Annual Symposium on Foundations of Computer Science, New Brunswick, NJ, USA, 20–23 October 2012; pp. 514–523.
20. Lee, W.L.J.; Verbeek, H.M.W.; Munoz-Gama, J.; van der Aalst, W.M.P.; Sepúlveda, M. Replay using recomposition: Alignment-based conformance checking in the large. In Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Management (BPM 2017), Barcelona, Spain, 13 September 2017.
21. Aggarwal, C.C.; Philip, S.Y. A survey of uncertain data algorithms and applications. *IEEE Trans. Knowl. Data Eng.* **2008**, *21*, 609–623. [CrossRef]
22. Suciu, D.; Olteanu, D.; Ré, C.; Koch, C. Probabilistic databases. *Synth. Lect. Data Manag.* **2011**, *3*, 1–180. [CrossRef]
23. Chui, C.K.; Kao, B.; Hung, E. Mining frequent itemsets from uncertain data. In Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, Nanjing, China, 22–25 May 2007; pp. 47–58.
24. Al-Mutawa, H.A.; Dietrich, J.; Marsland, S.; McCartin, C. On the shape of circular dependencies in Java programs. In Proceedings of the 2014 23rd Australian Software Engineering Conference, Milsons Point, Australia, 7–10 April 2014; pp. 48–57.
25. Bayes, T. LII. An Essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F.R.S. communicated by Mr. Price, in a letter to John Canton, A.M.F.R.S. *Philos. Trans. R. Soc. Lond.* **1763**, *53*, 370–418.
26. Mariappan, M.; Vora, K. GraphBolt: Dependency-Driven Synchronous Processing of Streaming Graphs. In Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, 25–28 March 2019; p. 25.
27. Aho, A.; Lam, M.; Sethi, R.; Ullman, J.; Cooper, K.; Torczon, L.; Muchnick, S. *Compilers: Principles, Techniques and Tools*; Addison Wesley: Boston, MA, USA, 2007.
28. Mokhov, A.; Carmona, J.; Beaumont, J. Mining conditional partial order graphs from event logs. In *Transactions on Petri Nets and Other Models of Concurrency XI*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 114–136.
29. Lu, X.; Fahland, D.; van der Aalst, W.M.P. Conformance checking based on partially ordered event data. In *International Conference on Business Process Management*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 75–88.

30. Lu, X.; Mans, R.S.; Fahland, D.; van der Aalst, W.M.P. Conformance checking in healthcare based on partially ordered event data. In Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, 16–19 September 2014; pp. 1–8.
31. Genga, L.; Alizadeh, M.; Potena, D.; Diamantini, C.; Zannone, N. Discovering anomalous frequent patterns from partially ordered event logs. *J. Intell. Inf. Syst.* **2018**, *51*, 257–300. [CrossRef]
32. Van der Aa, H.; Leopold, H.; Weidlich, M. Partial order resolution of event logs for process conformance checking. *Decis. Support Syst.* **2020**, *136*, 113347. [CrossRef]
33. Leemans, S.J.J.; Polyvyanyy, A. Stochastic-Aware Conformance Checking: An Entropy-Based Approach. In *International Conference on Advanced Information Systems Engineering*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 217–233.
34. Rogge-Solti, A.; van der Aalst, W.M.P.; Weske, M. Discovering stochastic petri nets with arbitrary delay distributions from event logs. In *International Conference on Business Process Management*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 15–27.