Temporal Mining for Interactive Workflow Data Analysis

Michele Berlingerio Fabio Pinelli Mirco Nanni Fosca Giannotti
ISTI - CNR Pisa, Italy
{name.surname@isti.cnr.it}

ABSTRACT

In the past few years there has been an increasing interest in the analysis of process logs. Several proposed techniques, such as workflow mining, are aimed at automatically deriving the underlying workflow models. However, current approaches only pay little attention on an important piece of information contained in process logs: the timestamps, which are used to define a sequential ordering of the performed tasks. In this work we try to overcome these limitations by explicitly including time in the extracted knowledge, thus making the temporal information a *first-class citizen* of the analysis process. This makes it possible to discern between apparently identical process executions that are performed with different transition times between consecutive tasks.

This paper proposes a framework for the user-interactive exploration of a condensed representation of groups of executions of a given process. The framework is based on the use of an existing mining paradigm: Temporally-Annotated Sequences ($\mathcal{TAS}$). These are aimed at extracting sequential patterns where each transition between two events is annotated with a typical transition time that emerges from input data. With the extracted $\mathcal{TAS}$, which represent sets of possible frequent executions with their typical transition times, a few factorizing operators are built. These operators condense such executions according to possible parallel or possible mutual exclusive executions. Lastly, such condensed representation is rendered to the user via the exploration graph, namely the Temporally-Annotated Graph ($\mathcal{TAG}$).

The user, the domain expert, is allowed to explore the different and alternative factorizations corresponding to different interpretations of the actual executions. According to the user choices, the system discards or retains certain hypotheses on actual executions and shows the consequent scenarios resulting from the coresponding re-aggregation of the actual data.

KDD'09, June 28–July 1, 2009, Paris, France.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining

General Terms

Algorithms

Keywords

Workflow mining, temporal sequence mining

1. INTRODUCTION

In the past few years, many organizations have started to use information systems to support the execution of their business processes [16]. With the increasing number of these available systems, the volume of the available collected processes logs is growing rapidly. These logs are very useful in several fields: in design and production processes, it could be important to detect the actual state of the process, how many items have been produced and in how much time; in logistics, the optimization of times is crucial; every step should be made strictly on time, and if there are anomalies or problems, the entire logistic solution should be redesigned.
For such reasons, the interest in analysing process logs has been increasing rapidly in the last years [23, 21, 6]. However, such logs are hard to analyse from different points of view because there is too much data, the original process diagram is too complex, and there are too many users to observe. Several techniques, such as workflow mining, have been proposed to automatically derive the workflow models originating from the process logs [22, 25, 11]. Their focus is to derive the process model that was actually followed, and this can be different from the original one in several ways, e.g., certain tasks from the original process were never performed or were performed too many times, or the tasks performed were not in the original diagram. In addition, these techniques answer questions such as:

- Given the logged traces, what is the workflow network?
- Is the mined workflow network equivalent to the original design? (Delta Analysis)
- Is the mined workflow network better than the original design? (Performance Analysis)

However, current approaches mainly use the temporal information contained in the logs just for keeping track of the temporal order of the performed tasks.

Indeed, the temporal information associated with logs in the form of timestamps conceals knowledge that allows to distinguish among different temporal behaviours.

For example, suppose we have to execute tasks A, B and C and that the transition time from A to B is usually 1 minute, and from B to C it is 9 minutes. If we have a transition time of 9 minutes from A to B and 1 minute from B to C, we are in the presence of an anomaly during the process, even if the sequence of the performed tasks follows the process workflow. In this case, the usual workflow mining techniques do not detect the anomaly and therefore treat the abnormal execution as normal. In addition to anomaly detection, it could be useful to highlight situations in which some users are faster (or slower) than others in performing certain tasks, or situations in which some resources take too much time to be performed. In this sense, the model returned by the analysis process might be even richer than the original model, since temporal features of the tasks are often kept out of the design phase, or at least they are not explicitly specified in the model.

The contributions of this paper can be summarized in 3 points: (i) a mining method that extensively takes into consideration the time information, i.e., the extracted patterns representing a group of executions of a given process with similar execution times; (ii) extracted patterns are summarized by taking into account the semantics of the possible executions, namely parallelism or mutual exclusion; (iii) users can interact with the extracted and summarized patterns and explore alternative cases proposed by the system.

The first point is based on Temporally-Annotated Sequences ($\mathcal{TAS}$) mining, a novel mining paradigm equipped with an efficient algorithm proposed in [7, 9] and recently successfully applied to biological data [3, 4]. $\mathcal{TAS}$ are sequential patterns where each transition between two events is annotated with a typical transition time that is found frequently in the data. In principle, this form of pattern is useful in several contexts: for instance, (i) in web log analysis, different categories of users (experienced vs. novice, interested vs. uninterested, robots vs. humans) might react in similar ways to some pages - i.e., they follow similar sequences of web access - but with different reaction times; (ii) in medicine, the relationship in time between the onset of patients' symptoms, drug consumption, and response to treatments; (iii) in workflow logs, the typical data is a sequence of operations performed with specific moments. From this, it could be interesting to extract frequent sequences containing frequent temporal annotations. $\mathcal{TAS}$ patterns have been also used as building blocks for a truly spatio-temporal trajectory pattern mining framework [8]. In all these cases, enforcing fixed time constraints on the mined sequences is not a solution. It is desirable that typical transition times, when they exist, emerge from the input data.

In summary we propose a methodology for helping the domain expert in the analysis of process logs. This methodology aims at understanding which possible models might have generated such logs, and whether such models might also contain temporal constraints. The methodology can be broken down in 4 main pieces described later in section 4.

This framework has been applied to a real-world system: a manufacturing company. We collected the logs of 3 million transactions on 9 tasks for a total of about 1 million performed tasks processing the access to the design of various mechanic components to be put into production. This factory is located worldwide and therefore the tasks are executed by different users at different locations. The results are encouraging, and indeed unexpected behaviours emerge.

The rest of the paper is organized as follows. Section 2 summarizes work related to workflow mining and $\mathcal{TAS}$. Section 3 introduces the technical details of the $\mathcal{TAS}$ paradigm and how it can be used for mining time-annotated data. Section 4 is the core of the paper that presents the original contributions of our work. It describes the overall methodology: the formal definition of the *factorization* operators, the exploration graph $\mathcal{TAG}$, and the algorithm for the interactive workflow analysis. Section 5 presents a case study in which we applied the framework to a real dataset of process logs. Section 6 summarizes the contributions and the results of this paper and draws some future lines of research.

2. RELATED WORK

In this section we summarize some literature strictly relevant to the topics of this paper. We first start with works related to workflow mining, and then we present papers where the $\mathcal{TAS}$ mining paradigm is applied to different contexts.

2.1 Workflow Mining

In the past few years, research has been performed on discovering a process model from a set of process instances. Most of them assume the existence of a process model underlying the given set of process instances. Several different approaches have been proposed to solve such a problem: in [2, 13] directed graphs are used, and in both papers, the researchers consider tasks that can be executed in parallel. However, in [13] the notion of parallelism between tasks is more sophisticated, i.e. they go beyond the simple temporal dependency between tasks that was treated in [2]: they define *overlapping* and *disjointed* activities. Finite state machines are proposed in [5], and Petri-nets are used in [24] for representing process instances. In [12] the authors define the concept of *temporal graphs*, which help in modeling the dependencies among the performed tasks during specific instances of the processes. They propose three different algorithms that work with temporal graphs, itemsets or sequences. These algorithms solve the *temporal pattern discovery* problem, defined as the discovery of the maximal temporal graphs among all frequent temporal graphs. In this work, the authors consider the starting and ending time of each task to explicitly detect situations of parallelism or choice between pairs of tasks. However, they do not look for frequent transition times or execution times of the tasks, i.e. they use the temporal dimension only to detect the temporal dependencies between tasks. In [10] the authors deal with the problem of mining *unconnected* patterns in workflows, i.e. detecting sets of activities that are frequently executed together and do not exhibit explicit dependent relationships. They present two different algorithms for solving the problem. This paper uses the concept of *frequency* of a pattern, which we employ as well.

For a survey on workflow mining please refer to [22, 25].

All of the above works explore only on the temporal dependencies among performed tasks. In this work we try to go further by looking for frequent subsequences of tasks that show temporal dependencies and additionally are executed with a similar transition duration.

2.2 $\mathcal{TAS}$-based Mining

In [4, 3] it is shown how it is possible to apply the $\mathcal{TAS}$ mining paradigm to medical data when its structure is a sequence of clinical observations taken at different times. In this context the temporal dimension of the data is a variable

that should be taken in account in the mining process and returned as part of the extracted knowledge. In these papers a real-world medical case study was reported in which the $\mathcal{TAS}$ mining paradigm was applied to such a data.

In [8], the authors introduce a novel spatio-temporal pattern that formalizes the idea of aggregate movement behaviour. In their approach a trajectory pattern is a sequence of spatial regions that, on the basis of the source trajectory data, emerge as frequently visited in the order specified by the sequence; in addition, the transition between two consecutive regions in this sequence is annotated with typical travel times that emerge from the input trajectories.

3. THE TAS MINING PARADIGM

Time in FSP (Frequent Sequence Patterns) is mainly considered (i) for the sequentiality that it imposes on events; (ii) as a basis for user-specified constraints, aimed to select an interesting subset of patterns; (iii) as a pruning mechanism to shrink the pattern search space and make computation more efficient. In all of these cases, time is not explicitly returned in the ouput as timestamps or timestamped intervals, although in some cases interval precedence and overlap is expressed [19, 26, 14, 17, 18, 15, 20].

The $\mathcal{TAS}$ mining paradigm, introduced in [9], tries to overcome such limitations, by defining a form of sequential patterns annotated with temporal information (or temporally-annotated sequences, $\mathcal{TAS}$ in short) that represent typical transition times between events in the sequence.

More formally:

DEFINITION 1 ($\mathcal{TAS}$). *Given a set of items $\mathcal{I}$, a* temporally-annotated sequence *of length $n > 0$, called n-$\mathcal{TAS}$ or simply $\mathcal{TAS}$, is a couple $T = (s, \alpha)$, where $s = \langle s_0, ..., s_n \rangle$, $\forall_{0 \leq i \leq n} s_i \in 2^{\mathcal{I}}$ is called the* sequence*, and $\alpha = \langle \alpha_1, ..., \alpha_n \rangle \in \mathbb{R}^n_+$ is called the* (temporal) annotation*. $\mathcal{TAS}$ will also be represented as follows:*

$$T = (s, \alpha) = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_n} s_n$$

Making use of this new form of pattern, the standard sequential pattern mining problem is redefined as the extraction of frequent $\mathcal{TAS}$, in the following way.

DEFINITION 2 (FREQUENT $\mathcal{TAS}$). *Given a set $\mathcal{D}$ of $\mathcal{TAS}$, a time threshold τ and a minimum support threshold σ, we define the* τ-support *of a $\mathcal{TAS}$ T as $supp_{[\tau,\mathcal{D}]}(T) = \mid \{T^* \in \mathcal{D} \mid T \preceq_\tau T^*\} \mid$ and say that T is* frequent *in $\mathcal{D}$, given a minimum support threshold σ if $supp_{[\tau,\mathcal{D}]}(T) \geq \sigma$.*

Such definition is based on a containment relation, $\preceq_\tau$, that extends the basic sequence containment relation by adding temporal constraints to the occurrences of the pattern. Such constraints essentially require that the temporal gaps in the occurrence be similar to the transition times in the $\mathcal{TAS}$, where *similar* means, in this context, to be equal up to a maximal deviation τ:

DEFINITION 3 (τ-CONTAINMENT ($\preceq_\tau$)). *Given a n-$\mathcal{TAS}$ $T_1 = (s_1, \alpha_1)$ and a m-$\mathcal{TAS}$ $T_2 = (s_2, \alpha_2)$ with $n \leq m$, and a time threshold τ, we say that T_1 is τ-contained in T_2, denoted as $T_1 \preceq_\tau T_2$, if and only if there exists a sequence of integers $0 \leq i_0 < ... < i_n \leq m$ such that:*

1. $\forall_{0 \leq k \leq n}.\ s_{1,k} \subseteq s_{2,i_k}$
2. $\forall_{1 \leq k \leq n}. \mid \alpha_{1,k} - \alpha_{*,k} \mid \leq \tau$

where $\forall_{1 \leq k \leq n}.\ \alpha_{*,k} = \sum_{i_{k-1} < j \leq i_k} \alpha_{2,j}$.

In this paper, we will make use of a software described in [9], named MiSTA, that extracts the complete set of frequent $\mathcal{TAS}$, and returns a concise representation of the following form:

$$s_0 \xrightarrow{[a_1,b_1]} s_1 \xrightarrow{[a_2,b_2]} \dots \xrightarrow{[a_n,b_n]} s_n$$

that can be read as: the sequence $s_0 \rightarrow \dots \rightarrow s_n$ appears frequently in the dataset, with typical transition times $t_1 \in [a_1, b_1]$ for the first transition, $t_2 \in [a_2, b_2]$ for the second one, and so on.

The software also allows to focus on contiguous occurrences, i.e., to consider only subsequences with no gaps in the support calculation of $\mathcal{TAS}$. This feature will be exploited in the experimental part of the paper.

4. A $\mathcal{TAS}$-BASED WORKFLOW MINING APPROACH

In this section we introduce a methodology for helping the domain expert in the analysis of process logs, aimed at understanding which possible models might have generated such logs, and whether such models might also contain temporal constraints. The methodology is composed of the following elements:

- a $\mathcal{TAS}$-based representation of the original log traces, that filters out noisy behaviours and detects temporal regularities. Such representation consists of a set of frequent $\mathcal{TAS}$;
- a set of operators for recognizing and *factorizing* two standard components of workflow models – i.e., parallelism and choice – from the $\mathcal{TAS}$, keeping trace of the temporal component;
- a graph summarization of a database of $\mathcal{TAS}$, to provide the user with an easy-to-grasp view of the data;
- an iterative and interactive procedure for exploring different and alternative *factorizations* of the same database of $\mathcal{TAS}$, potentially corresponding to different interpretations of the original input traces.

Performing these operations manually, by analyzing large quantities of information (such as 1 million of tasks performed as in our case study in section 5) is unfeasible and may not guarantee to discover what the domain expert or the workflow designer was looking for.

In the following, we start the discussion by summarizing the ultimate objective of this work, i.e., an interactive workflow analysis system. Then, for ease of presentation, we first describe the kind of data our analysis starts from (i.e., workflow traces) and define the above mentioned *factorization* operators over such data type. After that, the $\mathcal{TAS}$-based representation of the input data is briefly described, extending the factorization operators to the case of $\mathcal{TAS}$ and defining a graph summarization of sets of $\mathcal{TAS}$. Both the factorization operators and the graph representation will be the building blocks of the final analysis system, which is then described in more detail.

4.1 Problem setting: workflow analysis

One of the most important objectives in workflow analysis consists in reconstructing (part of) the workflow model that

has generated a given dataset of process execution traces. This sort of *reverse engineering* operation is often very useful for comparing the model derived from the traces with the original model that generated them. This kind of comparison might highlight some design mistakes, useless or redundant parts of the model or, in general, a usage of the model that differs from the intentions of its designer (e.g., containing the systematic adoption of actions that were originally meant to be exceptional measures).

Reconstructing the model underlying a set of process traces usually requires to make some guesses about the scheduled order of operations in the model, or whether some sets of actions were executed in parallel (*parallelism*) or they were executed as mutually exclusive alternatives (*choice*). The method proposed in this work tries to perform such a reconstruction in a step-by-step fashion, selecting (with the aid of the user) and isolating at each stage a single relation between actions, and iterating the process till all significant relations were caught. The whole process can be sketched as follows:

1: Represent the input set of process traces through a set of frequent $\mathcal{TAS}$;
2: **while** user does not stop the execution **do**
3: Compute a graph-based summary of the actual set of $\mathcal{TAS}$;
4: Detect the potential cases of *parallelism* and *choice* between pairs of actions within the actual set of $\mathcal{TAS}$;
5: Ask the user to choose a single case of *parallelism* or *choice* to factorize, or to backtrack;
6: **if** backtrack
then Return to the set of $\mathcal{TAS}$ preceding the last factorization step;
else Factorize the chosen relation between two actions (*parallelism* or *choice*), and update the set of $\mathcal{TAS}$ accordingly;

As we can see, the approach requires the interaction with the user, for choosing, among the several possible alternatives available at each step, the *factorization* that looks more promising. Performing such choice automatically would require to have a function that correctly evaluates the quality or utility of any alternative (i.e., any case of *parallelism* or *choice*) and selects the best one. To the best of our knowledge, the state-of-art of the field is still far from defining any function of this kind having a sufficiently wide applicability, therefore our solution demands this heavily domain-dependent evaluation to the user. The interaction with the user is facilitated by means of a graphical, graph-based, summarization of the set of $\mathcal{TAS}$ at hand, which provides a complementary, more readable view of the same data, that can help in choosing the next most interesting factorization step to perform, among those listed by the system. These aspects are detailed in Sections 4.6 and following ones.

4.2 The process workflow context

The digital traces collected during the re-iterated execution of a workflow process essentially have a sequential nature, and describe the ordered list of actions executed in each run, together with the agents who performed them and the date/time of execution:

Definition 4 (Workflow trace, Workflow log). *Let $\mathcal{A}$ be a finite set of actions and $\mathcal{U}$ a finite set of users. Then $\sigma = \langle (a_1, u_1, t_1)(a_2, u_2, t_2) \dots (a_n, u_n, t_n) \rangle$, where $a_i \in \mathcal{A}, u_i \in \mathcal{U}$ and t_i is a timestamp describing when the user u_i atomically performed a_i, is a* Workflow trace *or* Process instance. *A set $\mathcal{L}$ of workflow traces is a* Workflow log.

Therefore, a workflow log describes several runs (i.e., instances) of the same workflow process, each run being represented as a sequence of operations. An example of such a data can be found in Table 1 in Section 4.8, where two workflow traces (identified by the column "Inst.ID") are represented, each containing 4 actions (tasks) performed by a unique user at different times.

Basic applications of workflow log analysis focus on the sequences of actions performed in each trace, therefore disregarding the user identity and the temporal information, and representing each trace essentially as a sequence of items. For instance, the sample workflow log in Table 1 could be reduced to a set of two sequences: $\{ x \rightarrow a \rightarrow b \rightarrow c, \quad x \rightarrow b \rightarrow a \rightarrow c \}$.

4.3 Detecting parallelism and choice

As mentioned above, a typical workflow model can schedule the actions in several ways, including sequential execution (action a must be executed before b), parallel execution (a and b are launched together), and choice (only one between a and b is executed). A simple way to infer the presence of a parallelism or of a choice looking at a set of process instances, then, consists in locating possible evidences (or just clues) of such relations in the traces. On one hand, two actions invoked in parallel can appear in the traces in any order; on the other hand, two actions that form a choice can never appear one after the other. Following these basic ideas we define two relations between actions, that hold when the workflow traces suggest that a pair of actions might be executed in parallel or as a choice:

Definition 5 (Items relationships). *Let I be a set of items, and S be a set of sequences of items. Then, given $a, b, x \in I$, we define the relations $a \parallel_x b$ (read "a is parallel to b w.r.t. x") and $a \%_x b$ (read "a is in choice with b w.r.t. x") as follows:*

- $a \parallel_x b \quad \Leftrightarrow \quad \exists\, s, s' \in S$ *such that:* $(x \rightarrow a \rightarrow b \sqsubseteq s) \wedge (x \rightarrow b \rightarrow a \sqsubseteq s')$;
- $a \%_x b \quad \Leftrightarrow \quad \exists\, s, s' \in S$ *such that:* $(x \rightarrow a \sqsubseteq s) \wedge (x \rightarrow b \sqsubseteq s')$, *and* $\nexists\, s'' \in S : (a \rightarrow b \sqsubseteq s'') \vee (b \rightarrow a \sqsubseteq s'')$;

where $\sqsubseteq$ is the substring relation, i.e., $s \sqsubseteq s''$ iff all items of s appear in s' in the same order and in contiguous positions.

In the above definition, the relation between two items takes into consideration not only their relative positions in the input sequences, but also a limited form of *context*: both items (a and b) are preceded by a common item (x). This is a trade-off between more conventional relations defined in literature (e.g., [16]), mostly focused only on the items involved, and a more general approach that takes into consideration a larger number of items in the *past context* and a number of items also in the *future context*, i.e., situations like $x_1 \rightarrow \dots \rightarrow x_N \rightarrow a \rightarrow b \rightarrow y_1 \dots \rightarrow y_M$. In our case, essentially, we are considering $N = 1$ and $M = 0$.

Example 1 ($\parallel_x$). *If we have the sequences: $x \rightarrow a \rightarrow b$, $x \rightarrow b \rightarrow a$, then, according to Definition 5, we can write: $a \parallel_x b$. On the contrary, in the case of sequences: $x \rightarrow a \rightarrow b$, $y \rightarrow b \rightarrow a$ there is no parallelism under*

our definition, since each context (resp. x and y) leads to a distinct and coherent order of a and b. More standard definitions of parallelism do not consider the provenance *of subsequences $a \to b$ and $b \to a$, therefore they are mixed up and interpreted as a unique evidence of a parallelism.*

EXAMPLE 2 ($\%_x$). *If we have the sequences: $x \to a \to b$, $x \to b \to d$, then, according to Definition 5, we can write: $a\%_x b$. If we add the sequence $x \to b \to a$ to this example, $a\%_x b$ does not hold anymore, while now it holds $a \|_x b$.*

After defining which pairs of items/actions might potentially be in relation, we provide a function that lists all such relations, divided in parallelisms and choices:

DEFINITION 6 (PARALLELISM DETECTOR). *We define an unary operator $\mathcal{P}(S)$ that associates to a set of sequences S the collection of relations of parallelism contained in S, i.e., $\mathcal{P}(S) = \{(x, a, b) \mid a \|_x b \text{ in } S\}$.*

DEFINITION 7 (CHOICE DETECTOR). *We define an unary operator $\mathcal{C}(S)$ that associates to a set of sequences S the collection of relations of choice contained in S, i.e., $\mathcal{C}(S) = \{(x, a, b) \mid a\%_x b \text{ in } S\}$*

EXAMPLE 3 (DETECTORS). *Given a set of sequences $S = \{x \to a \to b \to c,\ x \to b \to a\}$, the following holds:*

- $P(S) = \{(x, a, b)\}$
- $C(S) = \{(b, a, c)\}$

The approach proposed in this work consists in iteratively selecting one of the possible relations between items, and then *factorizing* it in the traces, i.e., locating the occurrences of such relation and replacing the items involved with a new element that represents the pair of items and the relation that connects them. That yields a new set of traces, where the selected relation between items has been isolated and emphasized.

DEFINITION 8 (FACTORIZE$_\|$). *Let S be a set of sequences. Given $(x, a, b) \in \mathcal{P}(S)$, we define the operator $Factorize_\|((x, a, b), S) = S'$, where every subsequence $x \to a \to b$ or $x \to b \to a$ of $s \in S$ is replaced with $x \to a \parallel b$, where $a \parallel b$ is a new item.*

DEFINITION 9 (FACTORIZE$_\%$). *Let S be a set of sequences. Given $(x, a, b) \in \mathcal{C}(S)$, we define the operator $Factorize_\%((x, a, b), S) = S'$, where every subsequence $x \to a$ or $x \to b$ of $s \in S$ is replaced with $x \to a\%b$, where $a\%b$ is a new item.*

On the new set of traces obtained by applying one of the factorization operators above, the same kind of analysis (detection of relations) and transformation (factorization) can be applied, iteratively.

EXAMPLE 4 (FACTORIZATION). *Given S, $P(S)$ and $C(S)$ as in Example 3, we can apply the factorization operators in the following way:*

- $Factorize_\|((x, a, b), S) = S' =$
  $= \{\ x \to a \parallel b \to c\ ,\ x \to a \parallel b\ \}$
- $Factorize_\%((b, a, c), S) = S'' =$
  $= \{\ x \to a \to b \to a\%c\ ,\ x \to b \to a\%c\ \}$

4.4 A $\mathcal{TAS}$-based representation of traces

Applying the operators described above to the raw workflow traces has some drawbacks, mainly due to the possible presence or errors (missing actions, or actions registered by mistake) or very rare behaviours that we might want to exclude from the analysis.

Our approach provides that the analysis is carried out not on the original traces but on a set of $\mathcal{TAS}$ that represent the frequent behaviours (w.r.t. a given frequency threshold) and their temporal characteristics. That yields two results:

- first, errors and spurious behaviours are eliminated, since they are expected to appear with very low frequency, and therefore cannot appear in frequent patterns;
- second, the temporal information carried by the $\mathcal{TAS}$ can be used to better understand the behaviours appearing in the original traces, since different times in performing the same sequence of actions might reveal different usages of the same resources.

An example of $\mathcal{TAS}$ obtained from an input dataset of workflow traces is given in Table 2. Each $\mathcal{TAS}$ represents a sequence of actions (e.g., $x \to a$ in the first $\mathcal{TAS}$ listed) together with the set of typical transition times taken to move from one action to the next one (e.g., any time $t \in [18, 20]$, for the first $\mathcal{TAS}$).

The set of $\mathcal{TAS}$ used to represent the original traces can be selected following different criteria. Beside adopting different parameters and thresholds for the $\mathcal{TAS}$ mining phase, we could choose to use all the $\mathcal{TAS}$ extracted, or just the maximal ones, or those that satisfy other constraints, for instance temporal (e.g., take only patterns having duration longer than 5 minutes) or structural constraints (e.g., exclude patterns where the same action appears twice, thus evidencing the presence of a loop). For simplicity, in this paper we will adopt the first option, thus using all the $\mathcal{TAS}$ extracted. However, the whole analysis process can be equally applied with different selection criteria.

In the following we extend the operators described above in order to treat $\mathcal{TAS}$, instead of simple sequences.

4.5 Parallelism and choice over $\mathcal{TAS}$

All the definitions given for workflow traces do not take into account the temporal dimension contained in the data we work with. In order to add the time to our model, we redefine them for the case where the input sequences are a set of $\mathcal{TAS}$, as follows.

From now on, we assume to have a set of $\mathcal{TAS}$ T, each $\mathcal{TAS}$ being represented as a pair $t = (s, \alpha)$, where s is a sequence of items and α is a sequence of transition times. We also define as S_T the set of sequences that appear in T, without times, i.e., $S_T = \{s | (s, \alpha) \in T\}$. Then, definitions 5, 6 and 7 can be applied to S_T, essentially defining and locating parallelisms and choices only on the sequence component of our $\mathcal{TAS}$.

However, since when we solve a parallelism or choice instance we have to perform some operations to the temporal annotations on the corresponding sequences, we should redefine the factorization operators as follows.

DEFINITION 10 (FACTORIZE$_\|$). *Let T be a set of $\mathcal{TAS}$. Given $(x, a, b) \in \mathcal{P}(S_T)$, we define the operator $Factorize_\|((x, a, b), T) = T'$, where every temporally annotated substring $x \xrightarrow{\alpha_0} a \xrightarrow{\alpha_1} b$ of $(s, \alpha) \in T$ is replaced*

by $x \xrightarrow{\alpha_0} a \parallel b$*, and every temporally annotated substring* $x \xrightarrow{\alpha'_0} b \xrightarrow{\alpha'_1} a$ *of* $(s', \alpha') \in T$ *is replaced by* $x \xrightarrow{\alpha'_0} a \parallel b$*, where* $a \parallel b$ *is a new item.*

DEFINITION 11 (FACTORIZE$_\%$). *Let* T *be a set of* $\mathcal{TAS}$*. Given* $(x, a, b) \in \mathcal{C}(S_T)$*, we define the operator* $Factorize_\%((x, a, b), T) = T'$*, where every temporally annotated substring* $x \xrightarrow{\alpha_0} a$ *of* $(s, \alpha) \in T$ *is replaced by* $x \xrightarrow{\alpha_0} a\%b$*, and every temporally annotated substring* $x \xrightarrow{\alpha'_0} b$ *of* $(s, \alpha) \in T$ *is replaced by* $x \xrightarrow{\alpha'_0} a\%b$*, where* $a\%b$ *is a new item.*

EXAMPLE 5 (FACTORIZATION). *Given a set of frequent* $\mathcal{TAS}$ $T = \{x \xrightarrow{[18,20]} a \xrightarrow{[3,4]} b \xrightarrow{[7,10]} c,\ x \xrightarrow{[19,22]} b \xrightarrow{[2,4]} a\}$*, its corresponding set of sequences is* $S_T = \{x \to a \to b \to c,\ x \to b \to a\}$*. Then, we can apply the factorization operators in the following way:*

- $Factorize_\parallel((x, a, b), T) = T' = \{x \xrightarrow{[18,20]} a \parallel b \xrightarrow{[7,10]} c,\ x \xrightarrow{[19,22]} a \parallel b\}$
- $Factorize_\%((b, a, c), T) = T'' = \{x \xrightarrow{[18,20]} a \xrightarrow{[3,4]} b \xrightarrow{[7,10]} a\%c,\ x \xrightarrow{[19,22]} b \xrightarrow{[2,4]} a\%c\}$

4.6 A graph summarization of $\mathcal{TAS}$

The set of $\mathcal{TAS}$ extracted from a database of workflow traces can be rather large, though usually much less than the original data. That makes it difficult for a human expert to obtain an overall picture of the sequences of tasks described by the data by simply sifting through them. For this reason, in this work we define a graph data structure that provides a complementary, lossy yet easy-to-read view of the set of $\mathcal{TAS}$ under analysis.

DEFINITION 12 (TEMPORALLY-ANNOTATED GRAPH). *Given a set* T *of frequent* $\mathcal{TAS}$*, we define the* temporally-annotated graph ($\mathcal{TAG}$) for T *as a labeled graph* $G(T) = \langle V, E, l\rangle$*, whose nodes represent the actions appearing in* T*, the edges represent pairs of actions performed consecutively in at least one* $\mathcal{TAS}$ *of* T*, and the label of each edge is a set containing all the transition times that occurred in any* $\mathcal{TAS}$ *between the two corresponding consecutive actions. More formally:*

$$\begin{aligned} V &= \{a \mid a \sqsubseteq s, s \in S_T\} \\ E &= \{(a, b) \mid a \to b \sqsubseteq s, s \in S_T\} \\ l((a, b)) &= \{\alpha \mid a \xrightarrow{\alpha} b \sqsubseteq t, t \in T\} \end{aligned}$$

Figure 1 shows the Temporally-Annotated Graph corresponding to the starting set of $\mathcal{TAS}$ in Example 5. As we can see, all actions, all transitions between consecutive actions and all transition times contained in the $\mathcal{TAS}$ are depicted in the graph. On one hand, the graph loses some information, since all sequences longer than 2 in the $\mathcal{TAS}$ are virtually broken into pieces of length 2, not allowing to understand whether there is a loop in the starting sequences ($a \to b \to a$) or whether $b \to c$ is preceded by a in any sequence or, on the contrary, any sequence that passes through a terminates at b. On the other hand, the graph allows to understand at first sight some useful properties, for instance the fact that x plays the role of a source node, and c that of a terminal node, while between a and b there is not a strict order, which might be due to a loop or a case of parallelism. For comparison, in Figure 2 it is reported the $\mathcal{TAG}$ corresponding to the previous set of $\mathcal{TAS}$ after the factorization of a choice case. Notice that: (i) factorizing the choice case has as a side effect the disappearance of the parallelism located in the original set of $\mathcal{TAS}$, due to the fact that the two relations were in conflict and therefore the user must give priority to only one of them and disregard the other; (ii) the transition times for the rightmost edge ($b \to a\%c$) are obtained as union of those of $b \to a$ and $b \to c$, which is a direct effect of the way the labels of edges are defined in Definition 12.

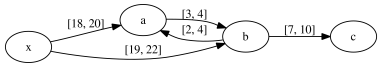

Figure 1: $\mathcal{TAG}$ for $\mathcal{TAS}$ T in Example 5

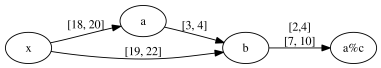

Figure 2: $\mathcal{TAG}$ after choice factorization in Examp. 5

Notice that our definitions of parallelism and choice involve a notion of *context*, that leads, in the case of parallelism, to check relations between actions in sequences of length 3 (which might become longer, if we extend the definitions to consider a longer context). That means that such relations cannot be clearly identified from the graph alone, and therefore the factorization analysis must be performed directly on the $\mathcal{TAS}$, as done in Section 4.5.

4.7 Interactive Workflow Analysis

The operators defined in the previous section allow to detect particular situations present in the dataset (parallelisms and choices), and to transform the latter in order to group the items involved.

We remark that the order of application of the operations is relevant, since after applying an operator the conditions for applying another operator could be not valid anymore (e.g., the result of a factorization for parallelism could destroy the subsequences that created a situation of choice), or simply the result could affect a different part of the dataset. In order to take into consideration all the possible sequences of operators applicable, we define a graph that represents the partially ordered set (*poset*) of all datasets that can be obtained from the original one (T), through a sequence of factorizations.

DEFINITION 13 (POSET GRAPH). *Given a dataset* T *of TAS, we represent the poset of transformations of* T *through a* poset graph $PG(T) = (V, E)$*, where:*

$$\begin{aligned} V &= PC^* \uparrow^\omega (\{T\}) \\ E &= \{(a, b) \in V \times V | b \in PC(\{a\})\} \end{aligned}$$

such that

$$\begin{aligned} PC(Ts) = {} & \{Factorize_\parallel((x, a, b), T) | \\ & \quad T \in Ts \wedge (x, a, b) \in \mathcal{P}(S_T)\} \\ \cup {} & \{Factorize_\%((x, a, b), T) | \\ & \quad T \in Ts \wedge (x, a, b) \in \mathcal{C}(S_T)\} \\ PC^*(Ts) = {} & Ts \cup PC(Ts) \end{aligned}$$

i.e., V *is the fix-point of operator* PC^* *applied to the original dataset, which yields the set of datasets obtained through a sequence of factorizations, and* E *connects each dataset with the dataset it was obtained from.*

If the original dataset of $\mathcal{TAS}$ is complex and contains critical situations, such that items involved in several parallelisms or choices, loops, etc., the set of transformed datasets can be very large. Therefore, it could be impractical for the end-user to simply fetch the whole graph of transformations. In Algorithm 4.7, we sketch an interactive procedure that extracts only a subset of the possible transformations, by asking the user which branch of the graph to explore, possibly backtracking to previous nodes of the graph: Figure 3 shows

Input: dataset of process logs L
Output: a set T of (factorized) $\mathcal{TAS}$
1: Extract the set $T = T_0$ of frequent $\mathcal{TAS}$ from L;
2: **while** execution not stopped by the user **do**
3: Compute the $\mathcal{TAG}$ on T and display it;
4: Compute the set $S = \mathcal{P}(S_T) \cup \mathcal{C}(S_T)$;
5: Present S to the user and ask him/her to select an operation *op* from $S \cup \{backtrack\}$;
6: **if** $op = backtrack \wedge T \neq T_0$ **then** $T = T'$ *s.t.* $(T', T) \in E$, $PG(T_0) = (V, E)$;
7: **else** $T =$ factorization of T through *op*;
8: **return** T;

Algorithm 1: Interactive Poset Graph Navigation

an example of a complete poset graph for a small dataset. The topmost $\mathcal{TAG}$ represents the (graph representation of the) set of $\mathcal{TAS}$ exctracted from the input workflow log, as described in steps 1–3 of Algorithm 1. Then, each arrow represents a possible factorization operation for a given set of $\mathcal{TAS}$ (see step 4), and each time the user chooses one of such operations (step 5) the algorithm factorizes the actual set of $\mathcal{TAS}$ accordingly, and re-iterates the computation focusing on the resulting set of $\mathcal{TAS}$.

4.8 Run-through example

In this section we present a run-through example on a toy dataset of only 2 days of logs, where each day represents a transaction. For each transaction we have a sequence of performed tasks, together with their timestamps. Table 1 shows the data under investigation. On this data we apply

Inst.ID	Task	User	Timestamp
1	x	Administrator	Oct, 09, 1980, 12:00:00
1	a	Administrator	Oct, 09, 1980, 12:00:19
1	b	Administrator	Oct, 09, 1980, 12:00:29
1	c	Administrator	Oct, 09, 1980, 12:00:31
2	x	User1	Oct, 10, 1980, 17:10:12
2	b	User1	Oct, 10, 1980, 17:10:13
2	a	User1	Oct, 10, 1980, 17:10:51
2	c	User1	Oct, 10, 1980, 17:10:54

Table 1: Example of Process Logs

the $\mathcal{TAS}$ mining paradigm, in order to extract sequences that are executed frequently with typical transition times. Table 2 shows the $\mathcal{TAS}$ mined with minimum support $\sigma = 10\%$ and temporal tolerance $\tau = 1$.

Figure 3 shows the poset graph of $\mathcal{TAG}$ that can be obtained starting from the $\mathcal{TAG}$ $G1$, which is the root of the graph. As we can see, we can have several possibilities at a certain level, for example after we generate graph $G2$. Each of them corresponds to having chosen to solve a particular parallelism or choice, first by enumerating all the possibilities by using one of the two *detector* operators defined in Section 4, then by applying the corresponding factorization operator. Choosing which parallelism or choice to solve will

$\mathcal{TAS}$ ID	$\mathcal{TAS}$	$\mathcal{TAS}$ ID	$\mathcal{TAS}$
1	$x \xrightarrow{[18,20]} a$	7	$x \xrightarrow{[18,20]} a \xrightarrow{[9,11]} b$
2	$x \xrightarrow{[0,2]} b$	8	$x \xrightarrow{[0,2]} b \xrightarrow{[37,39]} a$
3	$a \xrightarrow{[9,11]} b$	9	$a \xrightarrow{[9,11]} b \xrightarrow{[1,3]} c$
4	$a \xrightarrow{[2,4]} c$	10	$b \xrightarrow{[37,39]} a \xrightarrow{[2,4]} c$
5	$b \xrightarrow{[37,39]} a$	11	$x \xrightarrow{[18,20]} a \xrightarrow{[9,11]} b \xrightarrow{[1,3]} c$
6	$b \xrightarrow{[1,3]} c$	12	$x \xrightarrow{[0,2]} b \xrightarrow{[37,39]} a \xrightarrow{[2,4]} c$

Table 2: The corresponding mined $\mathcal{TAS}$

correspond to choose a path of $\mathcal{TAG}$ along the graph. In this way we can navigate through all the possible actions that we can perform on the original mined workflow $\mathcal{TAG}$.

5. CASE STUDY

In this section we present the work done as a case study on real-life data. The dataset comes from the usage of a real-world system developed by Think3[1], which is an object repository managing system, that allows the users to operate on the same objects from different locations. The timestamps contained on the logs represent the exact moment in which the event occurred. In particular, we did not have the starting and ending time of an operation, so we assumed that they are instantaneous and that the timestamps generally refer to the pair (execution time, transition time).

The dataset contains about 300000 transactions on 9 tasks, for a total of about 1 million of performed tasks. The logs span along 6 months of executions. For our analysis, we used a quite low support threshold of 0.5%, coupled with a τ of 1000 (seconds). Surprisingly, even these thresholds were enough to cut away two tasks from the frequently obtained annotated sequences. This proves that by manipulating the σ and τ parameters one can perform different grained analysis, even focusing on a frequently performed subprocess. Figure 4 shows a graph derived from the sequences of the original dataset of process logs in input, obtained with a procedure identical to the construction of $\mathcal{TAG}$, but without dealing with the temporal information.

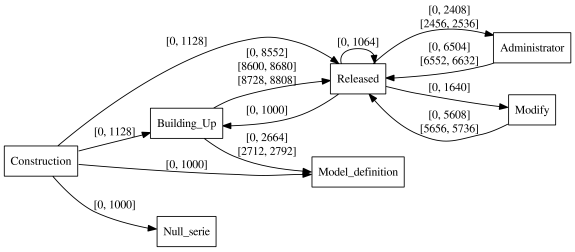

Figure 5: The initial mined $\mathcal{TAG}$

Figure 5 shows the $\mathcal{TAG}$ resulting from the initial mining step, before looking for any dependency among the activities. As we can see, the σ and τ parameter played already an interesting role in this first step: several paths in the graph have disappeared, making 2 out of 9 tasks disappear as well. Of course, using a lower minimum support and/or a higher τ would have resulted in a more selective mining, making several other paths and tasks disappear from the graph.

We then followed the steps we have described on the previous section: after running the $\mathcal{TAS}$ mining software, we applied all the operators we have in our framework, looking for interesting dependency situations. After one step

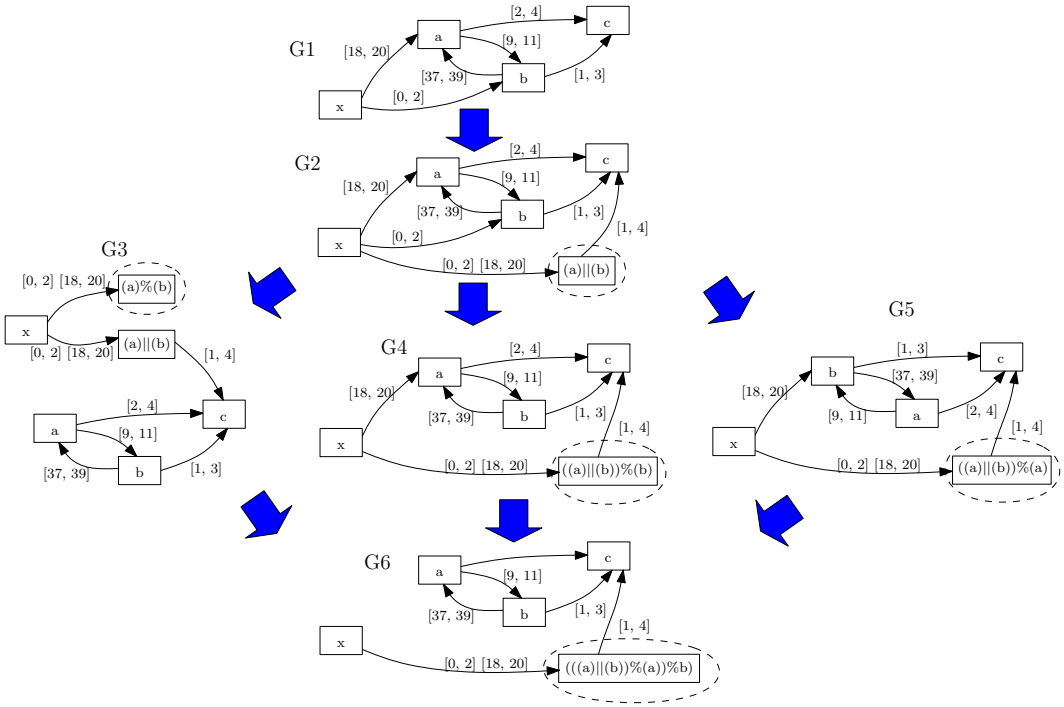

Figure 3: The poset of derived $\mathcal{TAG}s$ (dashed ellipses indicate the new items introduced by factorizations)

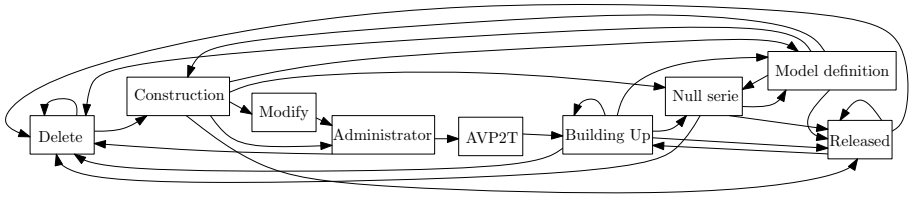

Figure 4: The graph derived from the original input data

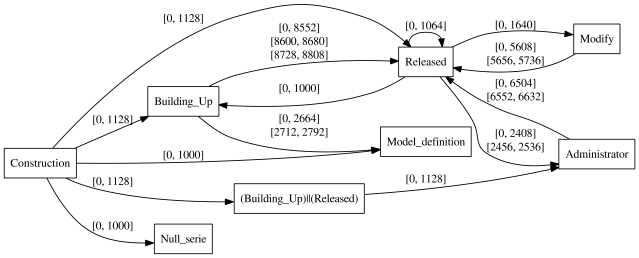

Figure 6: The $\mathcal{TAG}$ after one factorization step

of analysis, we found one parallelism and several choices. We followed the parallelism, obtaining the $\mathcal{TAS}$ graphically depicted in Figure 6.

If we go one step forward, solving the choice between *(Administrator)* and *(Modify)*, we can note an interesting event: due to the particular handling of the temporal annotations and to the definition of the choice splitter, the annotations of the *(Administrator)* task became split between the choice node and what was left to the old *(Administrator)* node. Thanks to this particular feature, it was possible to detect frequent temporal behaviours that can be actually divided in two sub-behaviours. This situation is depicted in Figure 7. As we can see, this framework is particularly suitable for any kind of temporal analysis of process logs. Thanks to the temporal annotations, it is easily possible to find bottlenecks on the process, unexpected behaviours, separate useless or redundant temporal information while performing business process analysis and so on. The $\mathcal{TAS}$ mining paradigm gives also the possibility, by a proper use of the minimum support parameter (σ), to look at the executed task with different granularity, looking for the most followed paths. The framework hence results particularly suitable for performing Delta Analysis and Performance Analysis. Analysts, in fact, can take advantage of our methodology in two ways: by using iteratively and interactively the two operators described in the paper, they can detect situations of choice and parallelism performed by the users (either as their free choice or because it was an intrinsic requirement of the corresponding tasks) that were not designed, and discover a workflow diagram different from the designed one (Delta Analysis); or they can take advantage of the temporal information contained in the $\mathcal{TAG}$ to discover bottlenecks or to optimize the execution of (part of) the process, by looking at the expected (possibly

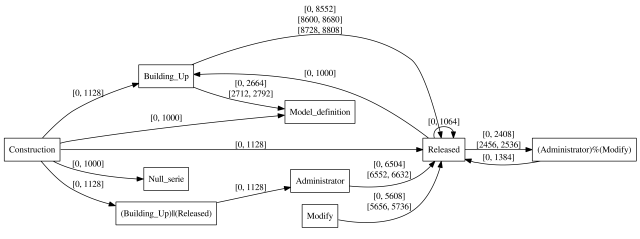

Figure 7: The $\mathcal{TAG}$ after two factorization steps

designed) time needed to perform particular (sequences of) tasks (Performance Analysis).

6. CONCLUSIONS AND FUTURE WORK

In this work we have introduced a novel framework for mining workflow graphs from process logs that enables the user to perform a temporal analysis by means of a $\mathcal{TAS}$-based mining paradigm. We have presented a methodology for helping the domain expert in the analysis of process logs, aimed at understanding which possible models might have generated such logs, and whether such models might also contain frequent temporal behaviours.

After a run-through example, we have presented a case study in which our model and framework have been used to perform visual temporal analysis on a real-life process log dataset. Based on this work, we have thus showed how the framework results suitable for performing Delta Analysis and Performance Analysis involving also the temporal dimension contained in the data. The results in these directions are encouraging, and indeed let emerge unexpected behaviours in our case study.

We plan to develop a complete software for performing such an analysis, which will guide the user through an iterative and interactive navigation of the poset of the possible workflow diagrams that the data can support. We plan also to investigate the possibility of extending the management of the transition times, in order to handle non-instantaneously executed tasks, which will enable an even more sophisticated temporal analysis of the data.

A possible research direction would be also to take the original designed workflow diagram as input, considering it during the mining step to better analyze the process logs.

7. REFERENCES

[1] The think3 company. http://www.think3.com.

[2] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. volume 1377-469+ of *LNCS*, '98.

[3] Michele Berlingerio, Francesco Bonchi, Fosca Giannotti, and Franco Turini. Mining clinical data with a temporal dimension: a case study. In *Proc. of The 1st Intern.Conf. on Bioinf. and Biomed.*, '07.

[4] Michele Berlingerio, Francesco Bonchi, Fosca Giannotti, and Franco Turini. Time-annotated sequences for medical data mining. In *Proc. of The Intern. Workshop of Data Min. in Medicine*, 2007.

[5] A. Datta. Automating the discovery of AI-IS business process models: probabilistic and algorithmic approaches. *Inf. Sys. Res.*, 9(3):275–301, '98.

[6] P. Lawrence (ed). *Workflow Handbook 1997, Workflow Management Coalition.* J. Wiley and S., NY, 1997.

[7] Fosca Giannotti, Mirco Nanni, and Dino Pedreschi. Efficient mining of temporally annotated sequences. In *Proc. of the 6th SIAM Intern. Conf. on Data Min.*, 2006.

[8] Fosca Giannotti, Mirco Nanni, Dino Pedreschi, and Fabio Pinelli. Trajectory patter mining. In *The 30th KDD Int.Conf. on Knowl.Disc. and Data Min.*, '07.

[9] Fosca Giannotti, Mirco Nanni, Dino Pedreschi, and Fabio Pinelli. Mining sequences with temporal annotations. In *Proc. of the 2006 ACM Symp. on Applied Comp. (SAC)*, pages 593–597, 2006.

[10] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, and Domenico Saccí. Mining unconnected patterns in workflows. *Inf. Syst.*, 32(5):685–712, 2007.

[11] Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng.*, 18(8):1010–1027, 2006.

[12] San-Yih Hwang, Chih-Ping Wei, and Wan-Shiou Yang. Discovery of temporal patterns from process instances. *Comput. Ind.*, 53(3):345–364, 2004.

[13] San-Yih Hwang and Wan-Shiou Yang. On the discovery of process models from their instances. *Decis. Support Syst.*, 34(1):41–57, 2002.

[14] Steffen Kempe and Jochen Hipp. Mining sequences of temporal intervals. In *PKDD*, pages 569–576, 2006.

[15] Frank Klawonn. Finding informative rules in interval sequences. In *Intelligent Data Analysis*, pages 123–132. Springer, 2001.

[16] Hongyan Ma. Process-aware information systems: Bridging people and software through process technology: Book reviews. *J. Am. Soc. Inf. Sci. Technol.*, 58(3):455–456, 2007.

[17] Fabian Moerchen. Algorithms for time series knowledge mining. In *Proc. of the 12th SIGKDD int.conf. on Knowl.disc. and data min.*, 2006.

[18] Panagiotis Papapetrou, George Kollios, Stan Sclaroff, and Dimitrios Gunopulos. Discovering frequent arrangements of temporal intervals. In *ICDM*, 2005.

[19] Dhaval Patel, Wynne Hsu, and Mong Li Lee. Mining relationships among interval-based events for classification. In *Proc. of the 2008 int.conf. on Manag.of data*, pages 393–404, 2008.

[20] Po shan Kam and Ada Wai chee Fu. Discovering temporal patterns for interval-based events. In *Proc. of the 2nd DaWaK*, pages 317–326. Springer, 2000.

[21] S.Jablonski and C.Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation.* Intern. Thomson Comp. Press, 1996.

[22] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.

[23] Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors. *Business Process Management, Models, Techniques, and Empirical Studies*, volume 1806 of *LNCS*. Springer, 2000.

[24] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT Press, 2002.

[25] T. Weijters and W. M. P. van der Aalst. Process mining: Discovering workflow models from event-based data., 2001.

[26] Edi Winarko and John F. Roddick. Discovering richer temporal association rules from interval-based data. In A. Min Tjoa and J. Trujillo, editors, *7th DaWaK*, volume 3589 of *LNCS*, pages 315–325. Springer, '05.