A Framework for Semi–automated Process Instance Discovery from Decorative Attributes

Andrea Burattin
Student Member, IEEE
Department of Pure and Applied Mathematics
University of Padua, Italy

Roberto Vigo
Research and Development Department
SIAV S.p.A.
Rubano, Italy

***Abstract*—Process mining is a relatively new field of research: is final aim is to bridge the gap between data mining and business process modelling. In particular, the assumption underpinning this discipline is the availability of data coming from business process executions. In business process theory, once the process has been defined, it is possible to have a number of instances of the process running at the same time. Usually, the identification of different instances is referred to a specific "case id" field in the log exploited by process mining techniques. The software systems that support the execution of a business process, however, often do not record explicitly such information. This paper presents an approach that faces the absence of the "case id" information: we have a set of extra fields, decorating each single activity log, that are known to carry the information on the process instance. A framework is addressed, based on simple relation algebra notions, to extract the most promising case ids from the extra fields. The work is a generalization of a real business case.**

I. Introduction

Process mining is a relatively young discipline emerged from the join of data mining with BPM [1]. The final goal of this discipline is the reconstruction of a process model, that points out a certain perspective of a given business process. Together with discovery, process mining provides also techniques for process conformance analysis (given a model, it tries to map the actual executions into the original model). It has a lot of real-world applications, such as the one described in [2].

The idea underpinning process mining techniques is that most business processes, that are executed with the support of an information system, leave traces of their activity executions and this information is stored in the so-called "log" files. The aim of process mining is to discover, starting from those logs, as much information as possible. Examples of reconstructed information are the control flow of the activities [3], or the network of social interactions between the originators involved in the process [4]. In order to perform such reconstructions, it is necessary that the log contains a minimum set of information. In particular, for all the recorded activities, there must be:

- the name of the activity;
- the name of the process the activity belongs to;
- the process case identifier (i.e. a field with the same value for all the executions of the same process instance);
- the time when the activity is performed;
- the activity originator (if available).

Typically, these logs are collected in MXML files [5] or, according to a new recent standard, in XES files [6], so that they can be analyzed using the ProM tool [7]. When process mining techniques are introduced in new environments, the data can be sufficient for the mining (or can even be more than the necessary, as in [8]) or can lack some fundamental information, as considered in this paper.

In the context of this work, two similar concepts are used in different ways: "process instance" and "case id". The first term indicates the logical flow of the activities for one particular instance of the process. The "case id" is the way the process instance is implemented: typically, the case id is a set of one or more fields of the dataset, whose values identify a single execution of the process. It is important to note that there can be many possible case ids but, of course, not all of them are going to recognize the actual process.

The final aim of this paper is to give a general framework for the selection of the "most interesting" case ids and then let the final decision to an expert user. The source of this work is a real business need encountered by Siav S.p.A. on logs coming from its document management solution.

Section II introduces the main issues related to the selection of the case id and the contexts where the problem arises. In Section III we recall other works addressing the question, and the differences from our paper are highlighted. Section IV presents the conditions that enable the use of our technique, which is described in Section V. Sections VI reports some notes about the implementation of the algorithms, the settings and results of our experiments. Finally, Section VII concludes and sketches a line for future work.

II. The Problem of Selecting the Case ID

As already introduced in the previous section, it is worthwhile observing that process mining techniques assume that all logs, used as input for the mining, come from executions of business process activities. In a typical environment, there is a formal process definition (it is not required to the model to be explicit) that is "implemented" and executed. All these executions are sequentially recorded into the log files; those act as input for the mining algorithms [3] and these will generate the mined models (possibly different from the original process models).

One of the fundamental principles that underpins the idea of "process modeling" is that a defined process can generate any number of concurrent instances, "running" at the same time. Consider, as an example, the process model defined (in terms of Petri Net) in Figure 1: it is composed of five activities. The process always starts executing A; then B; C and D can run in any order and, finally, E. We can have more than one instance running at the same time so, for example, at the time t we can observe any number of activities that are executed. Figure 2 shows three instances of the same process: the first is almost complete; the second is just started and the last one has just completed the first two activities.

In order to identify different instances, it is easy to figure out the need of an element that connects all the observations belonging to the same instance. This element is called "case identifier" (or "case id"). Fig. 2 represents it with different colors and with the three labels c_i, c_{i+1} and c_{i+2}.

A. Process Mining in New Contexts

There is a clear distinction between a process model and its implementation, and there exists a lot of software tools for supporting the application of a business process. Nonetheless, a lot of software systems concur to support the process implementation, but typically the information they provide is not explicitly linked to the process model because of the lack of a case id. This is mainly due to business reasons and software interoperability issues.

Consider, for example, a document management system (DMS): a tool which can store and manage digital documents (or images of paper documents). Those systems are widely used in large companies, public administrations, municipalities, etc. Of course, the documents managed with a DMS can be referred to processes and protocols (consider, for the example, the documents involved in supporting the process of selling a product). In this case, the set of documents managed by the DMS users leaves a trace of what happens at the "business process level", since they are a support for the process activities.

The idea presented in this paper is to exploit the information produced by such a support system, not limiting to DMSs, in order to mine the processes in which the system itself is involved. The nodal point is that such systems typically do not log explicitly the case id. Therefore, it is necessary to infer this information, that enables to relate the system entities (e.g. documents in a DMS) to process instances.

III. Related Work

The problem of relating a set of activities to the same process instance is already known in literature. In [9], Ferreira and Gillblad presented an approach for the identification of the case id based on the Expectation-Maximization (EM) technique. The main characteristics of this approach are its generality, that allows to execute it in all possible environments, and its computational complexity; furthermore, the EM-based algorithms converge to a local maximum of the likelihood function.

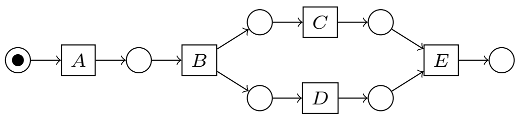
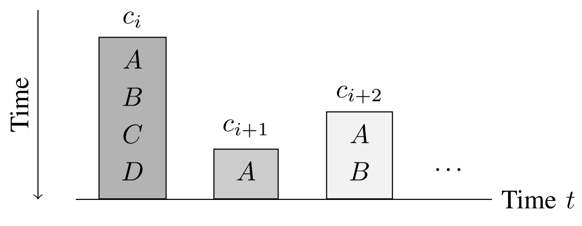


Figure 1. Example of a process model. In this case, activities C and D can be executed in parallel, i.e. in no specific order.



Time	Case	Activity
$t-3$	c_i	A
$t-2$	c_i	B
$t-1$	c_i	C
$t-1$	c_{i+2}	A
t	c_{i+1}	A
t	c_i	D
t	c_{i+2}	B

Figure 2. Two representations (one graphical and one tabular) of three instances of the same process (the one of Figure 1). It can be noticed that, at time t, the three executions of the same process reached different points.

Other approaches, such as the one presented by Ingvaldsen et al., in [10] and in [11], use the input and output produced by the activities registered in the SAP ERP. In particular, they construct "process chains" (composed by activities belonging to the same instance) by identifying the events that produce and consume the same set of resources. The assumption underpinning this approach (i.e. the presence of resources produced and consumed by two "joint" activities) seems too strong for a broad and general application.

Other works that can be reduced to the same issue are presented in [12], [13], but all of them solve only partially the problem and do not provide a sufficiently general approach that can be applied to other environments.

The most important difference between this work and others in literature is that, in our case, the information on the process instance is "hidden inside the log" (it is recorded in some fields we ignore), and therefore it has to be extracted properly. Such a difference is very important for two main reasons: *(i)* the settings we required are sufficiently general to be observed in a number of real environments and *(ii)* our technique is devised for this particular problem, hence it is more efficient than others.

IV. Working Framework

The process mining framework we address is based on a set $\mathcal{L}$ of *log entries* originated by auditing activities on a given system. Each log entry $l \in \mathcal{L}$ is a tuple of the form

$$(activity,\ timestamp,\ originator,\ info_1,\ \ldots,\ info_m)$$

being

- *activity* the name of the registered activity;
- *timestamp* the temporal instant in which the activity is registered;
- *originator* the agent that executed the registered activity;
- $\mathit{info}_1, \ldots, \mathit{info}_m$ possibly empty additional attributes. The semantics of these additional attributes is a function of the activity of the respective log entry, that is, given an attribute info_k and activities a_1, a_2, info_k entries may represent different information for a_1 and a_2; moreover, the semantics is not explicit. We call these data "decorative" or "additional" since they are not exploited by standard process mining algorithms. Observe that two log entries, referring to the same activity, are not required to share the values of their additional attributes.

Table I shows an example of such a log in a document management environment; please note the semantic dependency of attribute info_1 on activities: in case of "Invoice" it may represent the sender, in case of "Cash order" it may represent the account owner. The difference between such log entries and an event in the standard process mining approach is the lack of process instance information: bridging this gap, i.e. identifying the case id, is the target of our work. More in general, it can be observed that the source systems we consider do not implement explicitly some workflow concepts, since $\mathcal{L}$ might not come from the sampling of a formalized business process at all. Actually, $\mathcal{L}$ lacks also a process information: the issue is addressed in Section V-D.

In the following we assume a relational algebra point of view over the framework: a log $\mathcal{L}$ is a relation (set of tuples) whose attributes are *(activity, timestamp, originator,* $\mathit{info}_1, \ldots, \mathit{info}_m$*)*. As usual, we define the projection operator $\pi_{a_1,\ldots,a_n}$ on a relation R as the restriction of R to attributes $a_1, \ldots, a_n$ (observe that duplicates are removed by projection otherwise the output would not be a set). For the sake of brevity, given a set of attributes $A = \{a_1, \ldots, a_n\}$, we denote a projection on all the attributes in A as $\pi_A(R)$. Analogously, given an attribute a, a value constant v and a binary operation θ, we define the selection operator $\sigma_{a\theta v}(R)$ as the selection of tuples of R for which θ holds between a and v; for example, given $a = \mathit{activity}$, $v =$ "Invoice", and θ being the identity function, $\sigma_{a\theta v}(\mathcal{L})$ is the set of elements of $\mathcal{L}$ having "Invoice" as value for attribute *activity*. For a complete survey about relational algebra concepts refer to [14].

It is worthwhile to notice that relational algebra does not deal with tuples ordering, a crucial issue in process mining. This is not a problem, however, because *(i)* the log can be sorted whenever required and *(ii)* this work concerns the generation of a log suitable for applying process mining techniques (and not a process mining algorithm itself).

From now on, identifiers $a_1, \ldots, a_n$ will range over activities. Moreover, given a log $\mathcal{L}$, we define the set $\mathcal{A}(\mathcal{L}) = \pi_{\mathit{activity}}(\mathcal{L})$ (distinct activities occurring in $\mathcal{L}$). Finally, we denote with $\mathcal{I}$ the set of attributes $\{\mathit{info}_1, \ldots, \mathit{info}_m\}$.

As stated above, our efforts are concentrated on extracting flow of information from $\mathcal{L}$, that is, guessing the case id for each entry $l \in \mathcal{L}$, according to the following restrictions on the framework. Fixed a log $\mathcal{L}$, we assume that

1) given a log entry $l \in \mathcal{L}$, if a case id exists in l, then it is a combination of values in the set of attributes $PI \subseteq \mathcal{I}$ (i.e. *activity*, *timestamp*, *originator* do not participate in the process instance definition);
2) given two log entries $l_1, l_2 \in \mathcal{L}$ such that $\pi_{\mathit{activity}}(l_1) = \pi_{\mathit{activity}}(l_2)$, if PI contains the case id for l_1, then it also contains the case id for l_2 (i.e., process instance attributes set is fixed per activity); this is implied by the assumption that the semantics of additional fields is a function of the activity, as stated above.

V. Identification of Process Instances

From the basis we defined, it follows that the process instance has to be guessed as a subset of $\mathcal{I}$; however, since the semantics is not explicitly given, it cannot be exploited to establish correlation between activities, hence the process instance selection must be carried out looking at the entries $\pi_{\mathit{info}_i}(\mathcal{L})$, for each $i \in [1, |\mathcal{I}|]$. Nonetheless, since the semantics of $\mathcal{I}$ is a function of the activity, the selection should be performed for each activity in $\mathcal{A}(\mathcal{L})$, for each attribute in $\mathcal{I}$, resulting in a computationally expensive procedure. In order to reduce the search space, in the following Section some intuitive heuristics are depicted, that we applied successfully in our experimental environment (cf. Section VI). Then, in Section V-B, we show how to carry out the selection process over the resulting search space.

A. *Exploiting* a-priori *Knowledge*

Experts of the source domain typically hold some knowledge about the data, that can be exploited to discard the less promising attributes. Let a be an activity, and $\mathcal{C}(a) \subseteq \mathcal{I}$ the set of attributes candidate to participate in the process instance definition, with respect to the given activity. Clearly, if no *a-priori* knowledge can be exploited to discard some attributes, then $\mathcal{C}(a) = \mathcal{I}$.

The experiments we carried out helped us define some simple heuristics for reducing the cardinality of $\mathcal{C}(a)$, basing on

- assumptions on the data type (e.g. discarding timestamps);
- assumptions on the case id expected format, like average length upper and lower bounds, length variance, presence or absence of given symbols, etc.

It is worthwhile to notice that this procedure may lead to discard all the attributes info_i's for some activities in $\mathcal{A}(\mathcal{L})$. In the following we denote with $\mathcal{A}(\mathcal{C})$ the set of all the activities that overcome this step, that is $\mathcal{A}(\mathcal{C}) = \cup_{a \in \mathcal{A}(\mathcal{L})}\{a \mid \mathcal{C}(a) \neq \emptyset\}$. $\mathcal{A}(\mathcal{C})$ contains all the activities which have some candidate attribute, that is, all the activities that can participate in the process we are looking for.

Table I

AN EXAMPLE OF LOG EXTRACTED FROM THE DOCUMENT MANAGEMENT SYSTEM: THERE IS ALL THE BASIC INFORMATION (SUCH AS THE ACTIVITY, THE TIMESTAMPS AND THE ORIGINATOR), PLUS A SET OF INFORMATION ON THE DOCUMENTS (*info*$_1$ . . . *info*$_m$).

Activity	**Timestamp**	**Originator**	***info***$_1$	***info***$_2$	. . .	***info***$_m$
a_1 = Invoice	2010-06-02 12:35:47	Alice	A	2010-06-02	. . .	. . .
a_2 = Waybill	2010-06-02 12:36:18	Alice	A	B	. . .	. . .
a_3 = Cash order	2010-06-03 17:41:01	Bob	A	2010-06-03	. . .	. . .
a_4 = Carrier receipt	2010-06-04 09:12:28	Charlie	A	B	. . .	. . .
a_1	2010-06-05 08:45:12	Eve	B	2010-05-12	. . .	. . .
a_2	2010-06-06 07:21:02	Alice	B	A	. . .	. . .
a_3	2010-06-06 11:54:23	Bob	C	2010-02-20	. . .	. . .
a_4	2010-06-06 15:15:37	Charlie	B	A	. . .	. . .
a_1	2010-06-08 09:55:14	Bob	D	2010-03-30	. . .	. . .
a_2	2010-06-08 10:11:22	Bob	D	C	. . .	. . .
a_3	2010-06-09 16:01:28	Bob	C	2010-06-08	. . .	. . .
a_4	2010-06-09 18:45:09	Charlie	D	D	. . .	. . .

B. Selection of the Identifier

After the search space has been reduced and the set $\mathcal{C}(a)$ has been computed for each activity $a \in \mathcal{A}(\mathcal{L})$, we must select those elements of $\mathcal{C}(a)$ that participate in the process instance. The only information we can exploit in order to automatically perform this selection is the amount of data shared by different attributes. Aiming at modeling real settings, we fix a sharing threshold T, and we retain as candidate those subsets of $\mathcal{C}(a)$ that share at least T entries with some attribute sets of other activities. This threshold must be defined with respect to the number of distinct entries of the involved activities, for instance as a fraction of the number of entries of the less frequent one.

Let (a_1, a_2) be a pair of activities, such that $a_1 \neq a_2$ and let PI_{a_1} and PI_{a_2} the corresponding process instances field. We define the function S that calculates the shared values among them:

$$S(a_1, a_2, PI_{a_1}, PI_{a_2}) = |\pi_{PI_{a_1}}(\sigma_{activity=a_1}(\mathcal{L})) \cap \pi_{PI_{a_2}}(\sigma_{activity=a_2}(\mathcal{L}))|$$

Observe that, in order to perform the intersection, it must hold $|PI_{a_1}| = |PI_{a_2}|$. Using such function, we define the process instance candidates for (a_1, a_2) as:

$$\varphi(a_1, a_2) = \{(PI_{a_1} \in \mathcal{P}(\mathcal{C}(a_1)), PI_{a_2} \in \mathcal{P}(\mathcal{C}(a_2))) \mid S(a_1, a_2, PI_{a_1}, PI_{a_2}) > T\}$$

where $\mathcal{P}$ denotes the power set. Elements of $\varphi(a_1, a_2)$ are pairs, whose components are those attribute sets, respectively of a_1, a_2, that share a number of values greater than T (i.e. the cardinality of the intersection of PI_{a_1} and PI_{a_2} is greater than T). In the following, we denote with φ_a the set of all the candidate attribute sets for activity a, i.e. $\varphi_a = \{PI \mid \exists a_1 \in \mathcal{A}(\mathcal{C}), PI_{a_1} \in \mathcal{P}(\mathcal{C}(a_1)).(PI, PI_{a_1}) \in \varphi(a, a_1)\}$.

This formula figures out some candidate process instances that may relate two activities: it is worthwhile noticing, however, that our target is the correlation of a set of activities whose cardinality is in general greater than 2. Actually, we want to build a sequence $S = a_{S_1}, \ldots, a_{S_n}$ of distinct activities. Nonetheless, given activity a_{S_i}, there may be a number of choices for $a_{S_{i+1}}$, and then a number of process instances in $\varphi(a_{S_i}, a_{S_{i+1}})$. Hence, a number of sequences may be built. We call *chain* a finite sequence C of n components of the form $[a, X]$, being a an activity and $X \in \varphi_a$. Let us denote it as follows:

$$C = [a_1, PI_{a_1}], [a_2, PI_{a_2}], \ldots, [a_n, PI_{a_n}]$$

such that $(PI_{a_i}, PI_{a_{i+1}}) \in \varphi(a_i, a_{i+1})$, with $i \in [1, n-1]$. We denote with C_i^a the i-th activity of the chain C, and with C_i^{PI} the i-th PI of the chain C. Observe that a given activity must appear only once in a chain, since a process instance is defined by a single attribute set. Given a chain C ending with element $[a_j, PI_{a_j}]$, we say that C is *extensible* if there exists an activity $a_k \in \mathcal{A}(\mathcal{C})$ such that $(PI_{a_j}, X) \in \varphi(a_j, a_k)$, for some set $X \in \mathcal{P}(\mathcal{C}(a_k))$. Otherwise, C is said to be *complete*. Moreover, we say that an activity a occurs in a chain C, denoted $a \in C$, if there exists a chain component $[a, X]$ in C for some attribute set X. Since an activity can occur in more than one chain with different process instances, in some case we write $PI_{a_i,C}$ to denote the process instance of activity a_i in chain C. Finally, let $\mathcal{A}(C)$ denote the set of activities occurring in a chain C. The empty chain is denoted with $\perp$.

Given a chain C we define the average value sharing $S(C)$ among selected attributes of C as:

$$S(C) = \frac{\sum_{1 \leq i < n} S\left(C_i^a, C_{i+1}^a, C_i^{PI}, C_{i+1}^{PI}\right)}{n-1}$$

where n denotes the chain length.

All the possible complete chains on $\mathcal{L}$ are built according to Algorithms 1 and 2.

Algorithm 1 Build Chains

```
1: for all $a \in \mathcal{A}(\mathcal{C})$ do
2:     for all $PI \in \varphi_a$ do
3:         Extend Chain($[a, PI]$)
4:     end for
5: end for
```

Algorithm 1 calls Algorithm 2 for all the extensible chains of length 1. Observe that the pseudo code of Algorithms 1 and 2 builds also some chains which are permutations of one

Algorithm 2 Extend Chain

```
Require: a chain $C = [a_1, PI_1], ..., [a_{i-1}, PI_{i-1}]$
1: for all $a_i \in \mathcal{A}(\mathcal{C}) \mid a_i \notin C$ do
2:     for all $PI_i \in \varphi_{a_i} \mid (PI_{i-1}, PI_i) \in \varphi(a_{i-1}, a_i)$ do
3:         $C = C, [a_i, PI_i]$
4:         return Extend Chain$(C)$
5:     end for
6: end for
```

another (cf. Section V-D).

Example 1. *The following example highlights the bias introduced by the order of activities and attributes selection in building chains. Limiting to the first two attributes of the log given in Table I we can derive the following settings:*

- $\mathcal{A}(\mathcal{C}) = \{a_1, a_2, a_3, a_4\}$
- $\mathcal{C}(a_1) = \{\mathit{info}_1\}$
- $\mathcal{C}(a_2) = \{\mathit{info}_1, \mathit{info}_2\}$
- $\mathcal{C}(a_3) = \{\mathit{info}_1\}$
- $\mathcal{C}(a_4) = \{\mathit{info}_1, \mathit{info}_2\}$

Observe that field info_2 is discarded as a candidate attribute for activities a_1, a_3, since its values have the form of timestamps (cf. Section V-A). We compute now the values of function φ for the activities in $\mathcal{A}(\mathcal{C})$. Let $T = 1$ and consider $\varphi(a_1, a_2)$:

- $PI_{a_1} = \{\mathit{info}_1\}$*, since it is the only available candidate attribute;*
- PI_{a_2} *may be chosen among the sets* $\{\mathit{info}_1\}, \{\mathit{info}_2\}, \{\mathit{info}_1, \mathit{info}_2\}$.

We need to find those attribute sets of a_1, a_2 respectively, that share at least two values in the log (since $T = 1$). It is simple to see that the values of info_1 exactly overlap for both the activities, thus $(\{\mathit{info}_1\}, \{\mathit{info}_1\})$ is such a pair; another valid pair is $(\{\mathit{info}_1\}, \{\mathit{info}_2\})$, since

$$\begin{aligned} S(a_1, a_2, \{\mathit{info}_1\}, \{\mathit{info}_2\}) &= |\pi_{\mathit{info}_1}(\sigma_{activity=a_1}(\mathcal{L})) \cap \\ &\quad \pi_{\mathit{info}_2}(\sigma_{activity=a_2}(\mathcal{L}))| \\ &= 2 > 1 \end{aligned}$$

in fact

$$\pi_{\mathit{info}_1}(\sigma_{activity=a_1}(\mathcal{L})) = \{A,\ B,\ D\}$$

and

$$\pi_{\mathit{info}_2}(\sigma_{activity=a_2}(\mathcal{L})) = \{B,\ A,\ C\}$$

Follow the values of function φ for all the activities in $\mathcal{A}(\mathcal{C})$ and the respective values of S

- $\varphi(a_1, a_2) = \left\{ \begin{array}{c} (\{\mathit{info}_1\}, \{\mathit{info}_1\}), \\ (\{\mathit{info}_1\}, \{\mathit{info}_2\}) \end{array} \right\}$ *with $S = 3$ and $S = 2$, respectively;*
- $\varphi(a_1, a_3) = \emptyset$
- $\varphi(a_1, a_4) = \left\{ \begin{array}{c} (\{\mathit{info}_1\}, \{\mathit{info}_1\}), \\ (\{\mathit{info}_1\}, \{\mathit{info}_2\}) \end{array} \right\}$ *with $S = 3$ and $S = 3$, respectively;*
- $\varphi(a_2, a_3) = \{(\{\mathit{info}_2\}, \{\mathit{info}_1\})\}$ *with $S = 2$;*
- $\varphi(a_2, a_4) = \left\{ \begin{array}{l} (\{\mathit{info}_1\}, \{\mathit{info}_1\}), \\ (\{\mathit{info}_1\}, \{\mathit{info}_2\}), \\ (\{\mathit{info}_2\}, \{\mathit{info}_1\}), \\ (\{\mathit{info}_2\}, \{\mathit{info}_2\}), \\ (\{\mathit{info}_1, \mathit{info}_2\}, \{\mathit{info}_1, \mathit{info}_2\}) \end{array} \right\}$ *with $S = 3$, $S = 3$, $S = 2$, $S = 2$ and $S = 2$, respectively;*
- $\varphi(a_3, a_4) = \emptyset$.

Assume to start the chain building process from activity a_1 and, for the sake of simplicity, assume to restrict the log to activities a_1, a_2, a_3. The next activity a_i has to be chosen among those such that $\varphi(a_1, a_i) \neq \emptyset$, thus only a_2 is selectable. $\varphi(a_1, a_2)$ contains two pairs: suppose to select $(\{\mathit{info}_1\}, \{\mathit{info}_1\})$. The first selection step is finished and the chain is

$$C = [a_1, \{\mathit{info}_1\}], [a_2, \{\mathit{info}_1\}]$$

Then, in order to extend the chain, we must look for an activity $a_i \neq a_1, a_2$ such that $\varphi(a_2, a_i) \neq \emptyset$ **and** *there exists a pair in $\varphi(a_2, a_i)$ that contains $\{\mathit{info}_1\}$ as its first component. For such conditions an activity does not exist, the chain is complete. It is simple to see, however, that both choosing the process instances pair $(\{\mathit{info}_1\}, \{\mathit{info}_2\})$ for (a_1, a_2), or starting from a_3, would produce complete chains of length 3. Since the candidate attributes are computed on a statistical basis, the chain length is not the only significant parameter to take into account, but the example underlines the importance of building all the complete chains.* □

1) Match Heuristics: In computing the amount of data shared by two activities via function φ, heuristics approaches may help in modeling the complexity of a real domain. Actually, the comparison performed between values does not need to be an identity match, but rather a fuzzy match can be implemented. Guided by this basic heuristics, we can substitute the intersection operator in φ with an approximation of its, whose definition may be domain specific or not. Simple examples we tested in our experimental environment are:

- equality match up to X leading characters,
- equality match up to Y trailing characters,

and their combinations. In general it is possible to use a measure for string distance.

C. Results Organization and Filtering

In the previous Sections we shown how to compute a number of chains (i.e. a number of logs); in general, a domain expert is able of telling apart "good chains" from less reasonable ones, but this could be a demanding task. This section aims presenting the problem of comparing different chains. In order to address this issue, it is worthwhile to analyze a methodology that helps restricting the number of possible chains.

Generally, we can reject a chain in favor of another one if and only if the latter contains all the activities of the former, and it is either simpler or it supports a higher confidence. Example of parameters to take into account are:

- the number of attributes in the process instance of a chain component (recall that each component has the same number of process instance attributes): a chain that concerns less attributes may be considered simpler, thus preferable since more readable for a human agent;
- the cardinality of the value sharing between chain components ($S(\cdot)$): a chain whose share factor is higher gives higher confidence; this parameter could be tuned by a threshold.

Let $\mathcal{H}$ be the set of complete chains computed by Algorithms 1 and 2, without permutations. Given two chains

$$\begin{array}{l} C_1 = [a_1, PI_{a_1}], \ldots, [a_n, PI_{a_n}] \\ C_2 = [b_1, PI_{b_1}], \ldots, [b_m, PI_{b_m}] \end{array}$$

in the set $\mathcal{H}$, we define an ordering operator $\sqsubseteq$ as in Equation 1. $\sqsubseteq$ defines a reflexive, antisymmetric, and transitive relation over chains, hence $(\mathcal{H}, \sqsubseteq)$ is a partially ordered set [15]. For the sake of simplicity, in the above formulation we do not use any threshold to tune the value sharing comparison.

The ordering we defined strives to equip the framework with a notion of "best chains", i.e. those chains which it is worthwhile suggesting to a domain expert. The maximal elements of the poset are exactly those chains, as underlined in Example 2.

Example 2. *Recall the settings of Example 1. In this case the set $\mathcal{H}$ contains the complete chains listed below.*

$$\begin{array}{l}
C_1 = [a_1, \{\mathit{info}_1\}], [a_2, \{\mathit{info}_1\}], [a_4, \{\mathit{info}_1\}],\ S(C_1) = 3 \\
C_2 = [a_1, \{\mathit{info}_1\}], [a_2, \{\mathit{info}_1\}], [a_4, \{\mathit{info}_2\}],\ S(C_2) = 3 \\
C_3 = [a_1, \{\mathit{info}_1\}], [a_2, \{\mathit{info}_2\}], [a_3, \{\mathit{info}_1\}],\ S(C_3) = 2 \\
C_4 = [a_1, \{\mathit{info}_1\}], [a_2, \{\mathit{info}_2\}], [a_4, \{\mathit{info}_1\}],\ S(C_4) = 2 \\
C_5 = [a_1, \{\mathit{info}_1\}], [a_2, \{\mathit{info}_2\}], [a_4, \{\mathit{info}_2\}],\ S(C_5) = 2 \\
C_6 = [a_1, \{\mathit{info}_1\}], [a_4, \{\mathit{info}_2\}], [a_2, \{\mathit{info}_2\}], [a_3, \{\mathit{info}_1\}] \\
C_7 = [a_2, \{\mathit{info}_1, \mathit{info}_2\}], [a_4, \{\mathit{info}_1, \mathit{info}_2\}]
\end{array}$$

Applying the ordering relation we can compare different chains, as follows:

- $C_1 = C_2$*: in fact both* $C_1 \sqsubseteq C_2$ *nor* $C_1 \sqsupseteq C_2$ *holds;*
- C_1, C_2 *are not comparable with* C_3*, since* $\mathcal{A}(C_1) = \mathcal{A}(C_2)$ *neither contain nor are contained in* $\mathcal{A}(C_3)$*;*
- $C_1 = C_2 \sqsupset C_4$ *since* $S(C_1) = S(C_2) > S(C_4)$*;*
- $C_4 = C_5$*;*
- $\nexists i \in [1, 5] \cup \{7\}$ *such that* $C_i \sqsupset C_6$*;*
- C_7 *is not comparable with* C_3*.*

The Hasse diagram representing the poset $(\mathcal{H} \cup \{\perp\}, \sqsubseteq)$ *is given in Figure 3. It follows that only chain* C_6 *should undergo the experts' inspection.* □

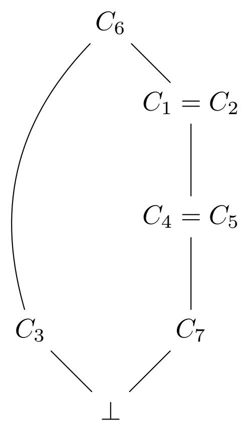


Figure 3. Hasse diagram representing the poset $(\mathcal{H} \cup \{\perp\}, \sqsubseteq)$ of Example 2.

D. Deriving a Log to Mine

For each chain C with positive length, we can build a log $\mathcal{L}'$ whose tuples have the form:

(activity, timestamp, originator, caseid, processid)

Please observe that the process instance we selected is a set of attributes, whereas a single one is expected by standard process mining techniques. Hence, a composition function k from a set of values to a single one is needed (e.g. string concatenation). Eventually, the log $\mathcal{L}'$ is obtained, starting from $\mathcal{L}$ with the execution of Algorithm 3.

Algorithm 3 Conversion of $\mathcal{L}$ to $\mathcal{L}'$

```
 1: $\mathcal{L}' \leftarrow \emptyset$
 2: $\mathit{chainno} \leftarrow 0$
 3: for all $C \in \mathcal{H}$ do
 4:     $\mathcal{L}_C \leftarrow \sigma_{\mathit{activity} \in \mathcal{A}(C)}(\mathcal{L})$
 5:     for all $l \in \mathcal{L}_C$ do
 6:         $\mathit{activity} \leftarrow \pi_{\mathit{activity}}(l)$
 7:         $\mathit{timestamp} \leftarrow \pi_{\mathit{timestamp}}(l)$
 8:         $\mathit{originator} \leftarrow \pi_{\mathit{originator}}(l)$
 9:         $\mathit{caseid} \leftarrow k\left(\pi_{PI_{\mathit{activity},C}}(l)\right)$
10:         $\mathit{processid} \leftarrow \mathit{chainno}$
11:         $\mathcal{L}' \leftarrow \mathcal{L}' \cup \{(\mathit{activity, timestamp, originator,}$
                       $\mathit{caseid, processid})\}$
12:     end for
13:     $\mathit{chainno} \leftarrow \mathit{chainno} + 1$
14: end for
```

Observe that the chain construction is sequential (i.e. it iteratively adds a component) and hence there is an implicit ordering among its elements. In order to build the log $\mathcal{L}'$, once all the chains are complete (no more extensible), it is possible to ignore the chains that are permutations of a given one. Thus, some chains computed by Algorithm 1 can be discarded.

It is worthwhile observing, however, that maximal elements in the poset represent different processes. A conservative approach compels us considering each maximal chain as defining a distinct process. The following example illustrates the reason why we chose this approach: let

$$C_1 = \ldots, [a_{i-1}, PI_{a_{i-1}}], [a_i, PI_{a_i}], [a_{i+1}, PI_{a_{i+1}}], \ldots$$

$$C_2 = \ldots, [b_{j-1}, PI_{b_{j-1}}], [b_j, PI_{b_j}], [b_{j+1}, PI_{a_{j+1}}], \ldots$$

be two maximal chains where $PI_{a_{i-1}} \neq PI_{b_{j-1}}$; $PI_{a_i} \neq PI_{b_j}$; $PI_{a_{i+1}} \neq PI_{b_{j+1}}$ and $a_i = b_j$. In other words, C_1

$$A \sqsubseteq B \Leftrightarrow \begin{cases} \left|A_1^{PI}\right| \geq \left|B_1^{PI}\right| & \text{if } \mathcal{A}(A) = \mathcal{A}(B) \;\wedge\; S(A) = S(B) \\ S(A) \leq S(B) & \text{if } \mathcal{A}(A) = \mathcal{A}(B) \;\wedge\; S(A) \neq S(B) \\ \mathcal{A}(A) \subseteq \mathcal{A}(B) & \text{otherwise} \end{cases} \tag{1}$$

and C_2 contain the same activity a_i but with different process instances. Considering C_1 and C_2 as belonging to the same process is not desirable, since it can lead to inconsistent control flow reconstruction. Hence, each maximal chain defines a process and the domain expert is in charge of recognizing if different chains belong to the same real process. During the conversion of $\mathcal{L}$ to the process log $\mathcal{L}'$, we assign as process id a chain counter.

VI. Experimental Results

As explained in the introduction, this paper is connected to a real business need, and it presents a generalization of a research activity carried out by Siav S.p.A. The existing implementation is limited to process instances constituted by a single attribute (e.g. $|PI_i| = 1$), due both to *a-priori* knowledge about the domain and computational requirements. In particular, all the preprocessing steps that reduce the search space are implemented as Oracle store procedures, written in PL/SQL. Then the chain building algorithms are implemented in C#. Moreover, for improving performances, we do not compute the heuristics on the whole log, but on a fraction of random entries of its.

We experimented our implementation on logs coming from a document management system; the source log is reduced to the form described in Section IV after undergoing some preprocessing steps. We applied the algorithms to real logs obtaining concrete results, validated by domain experts. In Table II the main information is summarized: please notice that the expert chains are always within the set of maximal chains (computed by the algorithm) because selected among the fists. Figure 4 shows how chains evolve when the log cardinality scales up: in particular, notice that the number of chains tends to increase, while the poset structure tears down the number of chains we presented to the domain experts. Figure 5 plots the processing time: it is a function of the log cardinality, of the number of activities in the log (i.e. the number of possible chain components), and of the number of decorative attributes (i.e. the number of possible ways of chaining two components).

The knowledge of the application domain gave us the opportunity to implement some heuristics, as explained in Sections V-A and V-B1. The following criteria were selected in order to reduce the search space:

- a candidate attribute must have a string type (e.g. we discard timestamps and numeric types, that in our case mostly represent prices);
- the values of a candidate attribute must fulfill these requirements:
  - maximum average length: 20 characters,

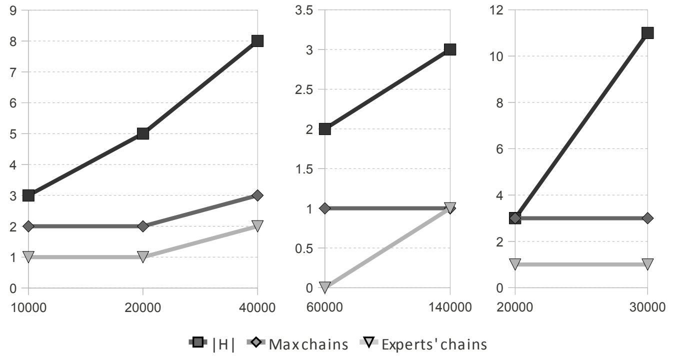


Figure 4. This figure plots the total number of chains the procedure identifies, the number of maximal chains and the number of chains the expert will select, given the size of the preprocessed log.

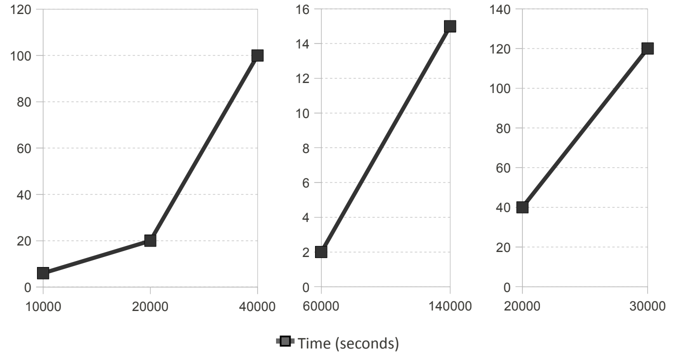


Figure 5. This figure represents the time (expressed in seconds) required for the extraction of the chains.

  - minimum average length: 3 characters,
  - maximum variation w.r.t. the average length: 10.

Finally, we relax the intersection operator in φ requiring values identity up to the first leading character and up to 2 trailing characters.

The experiments were carried out on an Intel Core2 Quad at 2,4 GHz, equipped with 4GB of RAM. The DBMS where the logs were stored was local to the machine, thus no network overhead has to be considered.

VII. Conclusion and Future Work

This work presents a new approach for the identification of process instances on logs generated from systems that are not process-aware. Process instance information is guessed exploiting additional information with respect to a standard process mining framework, but this information is typically available when dealing with software systems. Our work is a generalization of a real business case related to a document

Table II
RESULTS SUMMARY. HORIZONTAL LINES SEPARATE DIFFERENT LOG SOURCES.

$\lvert\mathcal{L}'\rvert$	$\lvert\mathcal{A}(\mathcal{L}')\rvert$	$\lvert\mathcal{I}\rvert$	**Time**	$\lvert\mathcal{H}\rvert$	**Maximal chains**	**Experts' chains**
10000	13	26	6 s	3	2	1
20000	39	26	20s	5	2	1
40000	47	26	1m 40s	8	3	2
60000	2	18	2 s	2	1	0
140000	4	18	15s	3	1	1
20000	12	16	40s	3	3	1
30000	16	16	2 m	11	3	1

management system, where discovering the process instance means correlating different document sets. The procedure we described to solve the problem is entirely based on the information that decorates documents, and relies on a relational algebra approach. Moreover, we deem that our generalization can be fairly adoptable in a number of domains, with a reasonable effort. Eventually, observe that we aim at applying some process mining techniques on a log computed on statistical basis, thus error-robust mining algorithms must be used.

There is still some important work that could improve the presented results. For example, we think it would be interesting to consider not only the value of the case id candidates, but to go deeper, to their semantic meaning (if any), which could act as *a-priori* knowledge. Moreover, a flexible framework for expressing and feeding the system with *a-priori* knowledge is desirable, in order to earn a higher level of generalization. Then, other refinements are domain-specific: dealing with documents, for instance, we could exploit their content in order to confirm or reject the findings of our algorithms, when the result confidence is low. Finally, before carrying on these new ideas, we are planning a wider test campaign on logs coming from other systems.

From the implementation point of view, it is desirable to extend the current version of the software in order to consider more than one attribute per process instance.

ACKNOWLEDGMENT

This work was entirely supported by SIAV S.p.A. We thank Nicola Costantini from SIAV and Prof. Alessandro Sperduti, for their important suggestions and supports.

REFERENCES

[1] W. M. P. van der Aalst, "Process Discovery: Capturing the Invisible," *IEEE Computational Intelligence Magazine*, vol. 5, no. 1, pp. 28–41, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5386178

[2] W. M. P. van Der Aalst, H. A. Reijers, A. J. M. M. Weijters, B. F. van Dongen, A. K. A. de Medeiros, M. Song, and H. M. W. Verbeek, "Business process mining: An industrial application," *Information Systems*, vol. 32, no. 5, pp. 713–732, Jul. 2007. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0306437906000305

[3] W. M. P. van Der Aalst, A. J. M. M. Weijters, B. F. van Dongen, J. Herbst, L. Maruster, and G. Schimm, "Workflow mining: a survey of issues and approaches," *Data & Knowledge Engineering*, vol. 47, no. 2, pp. 237–267, 2003.

[4] W. M. P. van Der Aalst and A. H. M. Ter Hofstede, "YAWL: yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245–275, Jun. 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0306437904000304

[5] W. M. P. van Der Aalst, B. F. van Dongen, C. W. Günther, R. S. Mans, A. K. A. de Medeiros, A. Rozinat, V. Rubin, M. Song, H. M. W. Verbeek, and A. J. M. M. Weijters, "ProM 4.0: Comprehensive Support for Real Process Analysis," in *Petri Nets and Other Models of Concurrency*, J. Kleijn and A. Yakovlev, Eds. Springer, 2007, pp. 484–494.

[6] C. W. Günther, "XES - Standard Definition," 2009. [Online]. Available: http://www.xes-standard.org/

[7] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst, "The ProM framework: A new era in process mining tool support," *Application and Theory of Petri Nets*, vol. 3536, pp. 444–454, 2005.

[8] A. Burattin and A. Sperduti, "Heuristics Miner for Time Intervals," in *European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2010.

[9] D. R. Ferreira and D. Gillblad, "Discovering Process Models from Unlabelled Event Logs," in *Business Process Management, 7th International Conference, BPM 2009*, ser. Lecture Notes in Computer Science, U. Dayal, H. A. Reijers, J. Eder, and J. Koehler, Eds., vol. 5701. Ulm, Germany: Springer, 2009, pp. 143–158.

[10] J. Espen Ingvaldsen and J. Atle Gulla, "Preprocessing Support for Large Scale Process Mining of SAP Transactions," in *Business Process Management Workshops*, A. H. M. Ter Hofstede, B. Benatallah, and H.-Y. Paik, Eds. Springer Berlin / Heidelberg, 2008, pp. 30–41. [Online]. Available: http://www.springerlink.com/content/4w3213725n4686hr

[11] ——, *Semantic Business Process Mining of SAP Transactions*, 1st ed. Business Science Reference, 2010, ch. 17, pp. 416–429.

[12] M. Walicki and D. R. Ferreira, "Mining Sequences for Patterns with Non-Repeating Symbols," in *IEEE Congress on Evolutionary Computation 2010*, Barcelona, Spain, 2010, pp. 3269–3276. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5585995

[13] D. R. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira, "Approaching Process Mining with Sequence Clustering : Experiments and Findings," in *Business Process Management*, G. Alonso, P. Dadam, and M. Rosemann, Eds., no. 1. Springer Berlin / Heidelberg, 2007, pp. 360–374.

[14] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 5th ed. Boston: Addison-Wesley, 2006.

[15] B. Schröder, *Ordered Sets: An Introduction*. Boston: Birkhäuser Boston, 2002.