Extracting Useful Knowledge from Event Logs: A Frequent Itemset Mining Approach

Youcef Djenouri, Asma Belhadi, Philippe Fournier-Viger

PII: S0950-7051(17)30486-0
DOI: 10.1016/j.knosys.2017.10.016
Reference: KNOSYS 4078

To appear in: Knowledge-Based Systems

Received date: 13 May 2017
Revised date: 12 October 2017
Accepted date: 14 October 2017


This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
Extracting Useful Knowledge from Event Logs: A Frequent Itemset Mining Approach

Youcef Djenouri¹, Asma Belhadi², Philippe Fournier-Viger³

¹ IMADA Lab, Southern Denmark University, Odense, Denmark
² Department of Computer Science, USTHB, Algiers, Algeria
³ Harbin Institute of Technology (Shenzhen), School of Social Sciences and Humanities, Shenzhen, China.
djenouri@imada.sdu.dk, abelhadi@usthb.dz, philfv8@yahoo.com

abstract Business process analysis is a key activity that aims at increasing the efficiency of business operations. In recent years, several data mining based methods have been designed for discovering interesting patterns in event logs. A popular type of methods consists of applying frequent itemset mining to extract patterns indicating how resources and activities are frequently used. Although these methods are useful, they have two important limitations. First, these methods are designed to be applied to original event logs. Because these methods do not consider other perspectives on the data that could be obtained by applying data transformations, many patterns are missed that may represent important information for businesses. Second, these methods can generate a large number of patterns since they only consider the minimum support as constraint to select patterns. But analyzing a large number of patterns is time-consuming for users, and many irrelevant patterns may be found. To address these issues, this paper presents an improved event log analysis approach named AllMining. It includes a novel pre-processing method to construct multiple types of transaction databases from a same original event log using transformations. This allows to extract many new useful types of patterns from event logs with frequent itemset mining techniques. To address the second issue, a pruning strategy is further developed based on a novel concept of pattern coverage, to present a small set of patterns that covers many events to decision makers. Results of experiments on real-life event logs show that the proposed approach is promising compared to existing frequent itemset mining approaches and state-of-the-art process model algorithms.

1. Introduction

Business processes are structured sets of activities performed using resources in an organization to achieve specific business goals. They are a key component of modern organizations and their management often determines the success of organizations in the long and in the short term. Business processes in modern organizations are supported by different types of information systems, which can collect information about their execution for monitoring and improvement purposes [18]. In other words, digitally supported business processes enable evidence-based business process manage-
To automate the analysis of business processes and exploit the huge amount of data collected about business processes, process mining has become crucial for many organizations. It consists of applying techniques to extract information about business processes from the logs of information systems supporting their execution [17]. These logs are usually referred to as event logs. While initially targeting the discovery of process models and the analysis of conformance between process models and event logs [1], process mining has grown into a larger set of event log analysis techniques to support different phases of the business process lifecycle. An important branch of process mining consists of applying data mining techniques to extract process knowledge deemed relevant to stakeholders [2, 24, 3].

Among these techniques, Frequent Itemset Mining (FIM) [27, 30] has attracted the attention of many researchers and practitioners in recent years as it can reveal frequent associations between resources and activities in event logs. According to this approach, a pattern is considered frequent, if its support (occurrence frequency) is no less than a user-specified minimum support threshold in an event log.

Although frequent itemset mining is widely used for process model extraction [10, 25, 24], current approaches based on frequent itemset mining have two important limitations. First, these methods are designed to be applied on original event logs. Because these methods do not consider other perspectives on the data that could be obtained by applying data transformations, many patterns are missed that may represent important information for businesses. Second, these methods can generate a large number of patterns since they only consider the minimum support as constraint to select patterns. Analyzing a large number of patterns is time-consuming for users, and many irrelevant patterns may be found.

These two limitations are illustrated with an example event log shown in Table 1, which depicts a small event log containing eight events partitioned into three traces. By applying frequent itemset mining on this event log, several patterns can be obtained.

- However a problem is that frequent itemsets only provide information about frequent associations between activities, resources and start times. For instance, the frequent itemset ("A, Pete", support = 25%) can be found, which indicates

<table>
<thead>
<tr>
<th>Trace</th>
<th>Event</th>
<th>CaseID</th>
<th>Activity</th>
<th>Resources</th>
<th>Start TimeStamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>tr1</td>
<td>e1</td>
<td>1</td>
<td>A</td>
<td>Mike</td>
<td>2015-08-21</td>
</tr>
<tr>
<td></td>
<td>e2</td>
<td>1</td>
<td>A</td>
<td>Pete</td>
<td>2015-08-25</td>
</tr>
<tr>
<td></td>
<td>e3</td>
<td>1</td>
<td>B</td>
<td>John</td>
<td>2015-08-28</td>
</tr>
<tr>
<td>tr2</td>
<td>e4</td>
<td>2</td>
<td>A</td>
<td>John</td>
<td>2015-08-29</td>
</tr>
<tr>
<td></td>
<td>e5</td>
<td>2</td>
<td>B</td>
<td>John</td>
<td>2015-09-23</td>
</tr>
<tr>
<td></td>
<td>e6</td>
<td>2</td>
<td>A</td>
<td>John</td>
<td>2015-09-24</td>
</tr>
<tr>
<td>tr3</td>
<td>e7</td>
<td>3</td>
<td>A</td>
<td>Pete</td>
<td>2015-09-25</td>
</tr>
<tr>
<td></td>
<td>e8</td>
<td>3</td>
<td>C</td>
<td>Mike</td>
<td>2015-09-28</td>
</tr>
</tbody>
</table>

Table 1: An Event Log
that the activity "A" is frequently performed by the resource "Pete", and the frequent itemset ("John", support = 50%) could be found, which means that the resource "John" is very active as it performed half of the activities for this business process. Although these patterns are interesting, many other interesting patterns are missed which could be found by analyzing the database using different perspectives or data transformations. For example, it would be relevant to answer the following questions for decision-makers: Which traces share many activities? Which traces share many resources? During which periods of the year the activity "A" is performed? When is the resource "John" used?

- The second problem is that if a low minimum support threshold value is used (e.g. 5%), many irrelevant frequent itemsets are extracted such as ("2015-08-21", support = 12.5%) and ("C, Mike", support = 12.5%). But if a high minimum support threshold value is used, many relevant frequent itemsets may be missing. For real-world application as in our context, the minimum support constraint is not enough to extract useful knowledge for the decision making process. Thus, additional constraint(s) should be considered to filter irrelevant patterns and show a small set of interesting patterns to the user that capture important aspects of the data.

To address these limitations of previous work, this paper proposes a novel frequent itemset mining approach for process mining called AllMining. The contributions of this work are threefold:

1. An elaborated pre-processing method is proposed to create multiple perspectives (transactional databases) on an event logs, which can be analyzed by frequent itemset mining algorithms to reveal different useful types of patterns. Several data transformation operations are proposed such as adding categorical attributes, and defining new types of databases from the perspectives of attributes and events. Thanks to the designed pre-processing approach, novel types of patterns are extracted by the proposed approach by applying an adapted FIM algorithm named SSFIM-EL on the transformed databases.

2. To find important process knowledge among the patterns extracted by frequent itemset mining, a filtering approach is proposed based on the concept of coverage. This approach can greatly reduce the number of itemsets presented to the user such that a small number of patterns can be used to support decision making.

3. An extensive analytical evaluation has been performed to compare the performance of the proposed approach with state-of-the-art process mining approaches using various performance measures such as runtime performance, memory consuming, and the number of frequent itemsets extracted. Furthermore, two case studies are discussed, describing the application of the proposed approach to analyze both lasagna and spaghetti processes. Our case studies reveal that our approach is particularly suitable to extract knowledge from event logs with high variability, where other techniques may fail to extract usable knowledge.

The reminder of this paper is organized as follows: Section 2 reviews studies on the use of data mining techniques for business process mining. Section 3 gives an
overview of the proposed approach, while Sections 4, 5 and 6 provide details about the
three steps of the proposed approach. Section 7 presents the performances evaluation.
Finally, section 8 draws a conclusion and discuss perspectives for future work.

2. Related Work

Several AI-based techniques have been proposed for process mining. Table 2 pro-
vides a classification of process mining approaches and their limitations.

The $\alpha$ algorithm [4] proposed by Van der Aalst et al. is the first process mining
algorithm. The authors prove that it can learn structured models from event logs by
defining sequences operators between activities using the footprint matrix. However,
the $\alpha$ algorithm is unable to discover structures such as short loops or non-
local, non-
free choice constructs. To address limitations of the $\alpha$ algorithm, several algorithms
have been proposed, to be used in different contexts, and using different techniques.

Several approaches have been designed using machine learning and evolutionary
computation techniques to learn declarative models. RNet [5] uses an Artificial Neural
Network (ANN) to analyze the behavior of customers. A drawback of this approach is
that neural networks are black box models, which are difficult to interpret by the end
user. Another approach to discover declarative models is the AGNES$^+$-miner [6]. It can
derive constraints that indicate whether an event can take place or not given previous
events. However, a limitation of this approach is that it is unable to discover complex
structures such as invisible and duplicate tasks. Therefore, an improved version called
AGNES$^+$-miner [7] was proposed by using a genetic algorithm. An efficient structure
called causal matrix was further introduced to facilitate search space exploration. An-
other algorithm for declarative process discovery is Declare-miner [8]. It lets users
guide the discovery process by specifying constraints that are converted to LTL (linear
temporal logic) formulas. Christian et al. [9] developed an adaptive fuzzy simplifica-
tion technique for discovering process models in an enactment log, which is based on
two metrics, namely significance and correlation.
Rule-induction techniques have also been used to extract process models. Maruster et al. [10] applied the RIPPER algorithm to derive rules between activities. The Inductive-miner algorithm [11] is employed to deal with complex-event log. It uses a divide-and-conquer strategy to recursively build a process model until all traces have been processed. For each trace, it first attempts to find regular sequences by dividing activities. It then defines adequate operators between the extracted activities. At each level of the recursion, one branch of the process tree is built, until the current event log becomes empty, i.e. all activities of all traces have been processed. Weijters et al. [12] presented an algorithm named HeuristicMiner for discovering process models in events log. However, this algorithm only considers the order of events within a case. The timestamp of an activity is used to calculate orderings between events. An improved version of this algorithm called FHM (Flexible Heuristic Miner) was proposed [13]. It is a flexible control-flow mining algorithm that performs well in practical situations and produce results that are easy to understand. Both algorithms can deal with noise and low frequency behavior, caused by the $\alpha$ algorithm. However, they are unable to discover large loops.

An important drawback of early approaches is their inability to extract rich models. To address this issue and motivated by the success of data mining techniques in many applications such as (urban traffic [14], agriculture [15], sentiment analysis [16] and so on), data mining-based approaches have been designed for process mining analysis. ActiTraC [17] is a clustering algorithm to improve the quality of the process model discovered. The resulting model is translated to procedural models, such as Petri-Nets.

Several approaches have also been proposed where high quality knowledge is extracted by Association Rule Mining (ARM). Huang et al. proposed to use association rule mining to discover resource allocation rules. A set of event log are transformed into transactional databases. Each item represents an activity and each transaction is a set of sequentially ordered activities representing the process case. Then, an ordered correlation among items is considered to represent the temporal issue released between activities in the event log.

The approach is implemented using radiology CT-scan examination process described in [20]. The results reveal that the approach gives useful resource allocation rules compared to Apriori-based algorithms.

Kamsu-Foguem et al. [21] proposed an ARM based approach for industrial process monitoring. The input data is created using three events of the Vam Drilling enterprise located in the south-west of France: i) Part size, containing the size of the objects that are being manufactured (small, medium and big), ii) Dysfunction occurrences, containing seven types of dysfunctions and iii) start-up delays, containing three types of real-time delays. After this step, association rules are extracted between these three types of events. This approach allows to improve production processes.

By applying ARM to process mining, a large number of rules can be extracted. This is an important issue since analyzing a large number of rules is time-consuming and difficult. Thus, other constraints should also be considered. Besides association rules, other types of patterns have also been extracted, providing different kind of information about business processes.

The Episode-miner algorithm [24] was proposed to discover frequent episodes in event logs. An episode is a partial ordering of events, which can be represented as a
An improved version of Episode-miner [25] was suggested by integrating an additional pruning strategy [26] to eliminate uninteresting frequent episodes. It was demonstrated that this algorithm is more efficient than the $\alpha$-algorithm in terms of the quality of the process model discovered.

An Apriori-based algorithm was proposed [22] to find frequent patterns in workflow logs. These patterns are then used to generate association rules for classifying new input events. A set of resource allocation constraints are employed by the proposed approach to filter patterns. The selected patterns are recommended to users for designing knowledge-based systems. An experimental evaluation has shown that the approach performed very well compared to traditional classifiers such as C4.5 and ANN in terms of recall and precision. However, the approach is very costly in terms of runtime. Maggi et al. [23] proposed a two-phase approach to improve the readability of discovered process models. In the first phase, the Apriori algorithm is applied to extract frequent activities from an event log. In the second phase, some measures are developed to prune frequent activities. The final set of frequent activities can be used to create constraints that describe the event log.

Frequent itemset mining approaches are useful for understanding event logs as they provide knowledge on associations between activities that can be easily interpreted by humans. However, these approaches still suffer from the problem of discovering a high number of patterns and of only considering the original event logs rather than applying pre-processing techniques to consider other perspectives on the event logs. To address these two issues, the next section proposes a novel process mining approach named AllMining.

### 3. Overall Framework

The proposed AllMining approach consists of three steps, depicted in Figure 1:

1. **Step1: Preprocessing.** This step consists of transforming the event log in a suitable form for analysis. First, the initial event log is transformed into a complete event log by adding some additional information. In the proposed approach, the user can create additional categorical attributes to address specific knowledge extraction objectives. Two types of categorical attributes are considered. Values of *basic* attributes are derived from information stored in single log events, while values of *complex* attributes are derived by combining information from multiple log events. For example, a *Quarter of the year* attribute where the value for an event is calculated using its timestamp is a basic attribute, while a *Direct Successor* attribute indicating the activity directly following an event, is a complex attribute as calculating each value requires information from two events. Second, from the complete event log, two types of transactional databases are created. Two strategies are proposed to attain this goal, each offering a different perspectives on an event log, to extract different types of patterns. The *event-based* strategy creates a transaction for each log event, while the *attribute-based* strategy creates a transaction for each attribute value of an event log.

2. **Step 2: Frequent itemset mining.** In this step, FIM techniques are applied to the transformed event log (obtained in Step 1) to extract frequent itemsets. It is to
Figure 1: General framework of the proposed approach
be noted that any frequent itemset mining algorithm can be applied in this step to extract frequent itemsets from the transformed event log. In the implementation of the designed approach, the SSFIM [31] algorithm is used to speed the mining process. To extract frequent itemsets, the user must specify a minimum support constraint, indicating in how many transactions each pattern must appear to be considered frequent.

3. **Step 3: Process Knowledge Discovery.** Applying FIM techniques in Step 2 can result in a very large number of itemsets. Analyzing a large number of itemsets is inconvenient for users. To address this issue, Step 3 aims here to derive from the set of all frequent itemsets, the set of knowledge readable and understandable by the user, by filtering irrelevant itemsets for decision-making. Thus, a novel heuristic pruning strategy is applied. This strategy is designed to maximize the number of log events covered by minimal set of frequent itemsets. This aims at preserving as much information as possible about the business process, while reducing the number of final patterns.

4. **Step 1: Preprocessing Step**

This first step consists of two phases: creating categorical attributes from the event log and constructing different transactional databases from the event log.

4.1. **Creating Categorical Attributes from the Event Log**

If an event log is analyzed as it is, the patterns discovered in the log may be useless or provide an incomplete view of the business process, as they would fail to consider important information that can be obtained by transforming the event log. To discover different types of rich patterns from an event log, this paper proposes to first create categorical attributes from the event log. The following types of attributes are considered:

1. **Basic information**: The first new type of attribute describes basic information about the event log and more specifically the duration of an activity and the time of the year where the activity was carried out. The *duration of an activity* in the event log is the time elapsed between the beginning and the end of the activity. The duration is expressed into hours and days. But it is then discretized into three categories (short, medium, and long duration). Note that it would be possible to use a smaller or greater number of categories, depending on the event log and the type of process model to be analyzed. The second information is the *quarter in which the activity has started and ended*. To encode this information, two attributes are added: Start Time of the Year and End Time of the Year. The first, second, third and fourth quarters are respectively defined as {January, February, March}, {April, May, June}, {July, August, September} and {October, November, December}.

2. **Complex information**: Besides basic information, categorical attributes are created to store complex information the relationships between activity and resources in the event log. In particular, categorical attributes are added to store
the precedence relationship between activity/resources in the event log. This relationship is based on the activity or resource that precedes or follows another activity or resource according to their start times. To create attributes encoding this relationship, the entire event log is scanned. For each process instance, all events are scanned. For each event, three pointers are used (previous, current and next). The preceding activity or resource of the current event is the activity or resource of the event pointed by the previous pointer. The following activity or resource of the current event is the activity or resource of the event pointed by the next pointer. If the next or previous pointer is null, the symbol ? is added to the Preceding/Following Activity or Resource of the current event.

Creating the above categorical attributes can be done by scanning all events once. Thus, the time complexity is \( O(M) \) where \( M \) is the number of events in the event log.

4.2. Constructing Multiple Transactional Databases From the Event log

In the second phase of the preprocessing step, multiple transactional databases are created from the event log to provide different perspectives to data analysts for discovering patterns in the log. In particular, two different strategies are proposed to create such perspectives. The first strategy, called the Events-Based Strategy (EBS), considers that each event is a transaction in a transaction database. The second strategy, called Attribute-Based Strategy (ABS), considers that each attribute is a transaction.

4.2.1. EBS: Events-Based Strategy

To be able to analyze an event log using FIM techniques, it is necessary to transform the event log into a transactional form. Whereas an event log contains categorical attributes, a transactional database contains Boolean items. The proposed approach transforms a categorical database into Boolean items by drawing inspiration from the work of Rastogi and Shim [36]. The following paragraphs explain how an event log is transformed into an EBS-database.

Assume that we have an event log \( E \) composed of \( m \) events \( \{e_1, e_2, ..., e_m\} \), and a set of attributes \( V = \{v_1, v_2, ..., v_k\} \), for describing events. Each attribute \( v_j \) has a set of possible values denoted as \( \text{dom}(v_j) = \{a_{j1}, ..., a_{jp}\} \). For each attribute \( v_j \), a function \( \text{EV}_j : E \rightarrow \text{dom}(v_j) \) assigns an attribute value to each event. The proposed EBS-database of an event log \( E \) is a set of \( m \) transactions \( \{t_1, t_2, ..., t_m\} \), where the \( i \)-th transaction \( t_i \) represents the \( i \)-th event \( e_i \) from the event log. The transaction \( t_i \) is defined as the union of the attribute values of the \( i \)-th event, that is \( t_i = \bigcup_{z=1}^{k} \text{EV}_z(e_i) \). The elements of a transaction are called items. The total number of items \( n \) in an EBS-database is \( n = \bigcup_{z=1}^{k} \text{dom}(v_i) \).

For instance, consider the event log described in Table 1. Table 3 presents the corresponding EBS-database of this event log instance by only considering the first trace, and only the Activity and Resources attributes.

4.2.2. ABS: Attributes-Based Strategy

The second proposed type of transformed database is the ABS-database. An ABS-database is created for each attribute, where each transaction represents an attribute value, and items in a transaction indicates the traces where that attribute value appears.
Consider the set of all traces $TR = \{tr_1, tr_2, ..., tr_q\}$ from an event log. For a trace $tr_i$, let the notation $Events(tr_i) = \{e_1^i, ..., e_r^i\}$ denotes the set of all events in the trace $tr_i$.

An attribute value $a$ of an attribute $v_j$ is said to be covered by a trace $tr$ (denoted as $a \models tr$) if there exists an event $e$ in $tr$ where $a$ appears for the attribute $v_j$. More formally, $a \models tr \iff \exists e \in dom(v_j) \land \exists e \in Events(tr) \land EV_j(e) = a$.

Consider an attribute $v_j$ having $p$ possible values $\text{dom}(v_j) = \{a_1^j, ..., a_p^j\}$. The ABS-database of $v_j$ is denoted as $T^{(j)}$. It consists of $p$ transactions $t_1, t_2, ..., t_p$, where $t_p = \{tr|a_j^p \models tr \land tr \in TR\}$. The elements of a transaction are called items. The total number of items $n$ in an ABS-database is the number of traces ($n = |TR|$).

For instance, consider the event log of Table 1. It contains three traces, i.e. $TR = \{tr_1, tr_2, tr_3\}$. The set of events is $E = \{e_1, e_2, ..., e_8\}$. The possible values for the Activity and Resources attributes are $\text{dom}(\text{Activity}) = \{A, B, C\}$ and $\text{dom}(\text{Resources}) = \{Mike, Pete, John\}$, respectively. The following relationships hold:

- $Mike \models (tr_1 \land tr_3)$ because Mike participates in ($e_1 \in tr_1$ and $e_3 \in tr_3$).
- $A \models TR$ because $A$ appears in all traces ($e_1, e_2$ for $tr_1$, $e_4, e_6$ for $tr_2$ and $e_7$ for $tr_3$).

To construct the ABS-database, the set of items $I$ is defined as the set of all traces, that is $I = TR = \{tr_1, tr_2, tr_3\}$. The first attribute is Activity. Thus, a transaction database $T^{(1)}$ is created for this attribute, where a transaction is created for each value of the Activity attribute. Since there are three values ($|\text{dom}(\text{Activity})| = 3$), $T^{(1)}$ contains three transactions:

- $t_1^{(1)}: tr_1, tr_2, tr_3$ which represents the activity $A$.
- $t_2^{(1)}: tr_1, tr_2$ which represents the activity $B$.
- $t_3^{(1)}: tr_3$ which represents the activity $C$.

Similarly, a transaction database $T^{(2)}$ is created for the second attribute Resources. This database contains three transactions, defined as follows:

- $t_1^{(2)}: tr_1, tr_3$ which represents the resource Mike.
- $t_2^{(2)}: tr_1, tr_3$ which represents the resource Pete.
- $t_3^{(2)}: tr_1, tr_2$ which represents the resource John.

<table>
<thead>
<tr>
<th>ID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>{A, Mike}</td>
</tr>
<tr>
<td>e2</td>
<td>{A, Pete}</td>
</tr>
<tr>
<td>e3</td>
<td>{B, John}</td>
</tr>
</tbody>
</table>

Table 3: The EBS-database corresponding to the database of Table 1
5. Step 2: Frequent itemset mining

The previous section has explained how the data is prepared in the proposed AllMining approach by pre-processing an event log to obtain several transaction databases. This section explains how AllMining extracts patterns from these databases to obtain key information about a business process. The section first briefly reviews how frequent itemsets are traditionally extracted from a transaction database. Then, it presents an adaptation of the recently proposed SSFIM [31] frequent itemset mining algorithm to discover frequent itemsets in multiple transactional databases constructed from an event-log, as defined in the previous section. Lastly, this section presents different types of knowledge that can be extracted from an event log using the proposed event log mining approach.

5.1. Traditional Frequent itemset mining

A transactional database $T$ is a set of transactions $\{t_1, t_2, \ldots, t_m\}$, where $I$ is the set of the $n$ different items or attributes $\{i_1, i_2, \ldots, i_n\}$ appearing in the database. An itemset $X$ is a set of items such that $X \subseteq I$. The support of an itemset $I' \subseteq I$ is the number of transactions containing $I'$ divided by $m$, the number of transactions in the database. A given itemset is called frequent if it appears in at least $m \times \text{minsup}$ times in $T$ transactions, where $\text{minsup}$ is a threshold set by the user. Frequent itemset mining is the task of extracting all frequent itemsets from a given transactional database[27].

For example, consider the transactional database depicted in Table 4. It contains five transactions $\{t_1, t_2, t_3, t_4, t_5\}$ and five items $\{A, B, C, D, E\}$. The support of the itemset $\{A, B\}$ is $2/5$ since it appears in two out of five transactions. If the minimum support is set to a value no less than $0.4$, then the itemset $\{A, B\}$ is frequent. Otherwise, it is infrequent.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>$t_4$</td>
<td>E</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>$t_5$</td>
<td>C</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: A Transactional Database

5.2. Mining Multiple Transactional Databases with the Modified SSFIM algorithm

In the last decades, numerous algorithms have been developed for discovering frequent itemsets such as Apriori [27], FP-Growth [28] and DIC [29]. However, these algorithms may have long execution times when applied on large transactional databases. In this paper, the multiple transactional databases generated from an event log are sparse. The EBS-database is created from an event log. Then, an ABS-database is created for each attribute of the initial event log. If there are $k$ attributes, then the total number of databases to be analyzed will thus be $k + 1$. Because these databases share several items, it is desirable to mine all these databases using the SSFIM algorithm.
(a recent algorithm designed to deal with sparse transactions). The main motivation to use SSFIM on event logs is that it was shown to very efficient for databases with a large number of transactions, where each transaction contains few items, which is the case of the databases in this paper.

Indeed, the SSFIM algorithm (Single Scan for Frequent itemset mining) is adopted in the proposed AllMining approach. The proposed algorithm called SSFIM-EL (Single Scan for Frequent itemset mining on event log) processes $k + 1$ transactional databases constructed in the previous step.

The aim of SSFIM-EL is to minimize the number of database scans and the number of candidates generated while discovering the frequent itemsets to overcome the limitations of classical FIM algorithms.

The principle guiding SSFIM-EL is to iteratively mine multiple transactional databases generated in the previous step. For each transactional database constructed, the transactions can thus be processed one by one to obtain candidate frequent itemsets. While processing transactions, the support of these candidate itemsets can be updated, so that at the end of a single database scan, it is possible to assess which itemsets satisfy the minsup constraint.

SSFIM-EL generates all possible itemsets from a transaction. If a generated itemset $t$ has already been generated when processing a previous transactions, its support is incremented by one. Otherwise, it is initialized to one. The process is repeated until all transactions have been processed.

The SSFIM-EL algorithm for mining transactional database generated from an event log is given in Algorithm 1. The input is a set $T$ of $d$ transactional databases and a minimum support value $\text{minsup}$. The output is the set of all frequent itemsets $F$. For each transactional database, the SSFIM-EL algorithm first creates an hash table $h$ to store all generated candidate itemsets together with their current number of occurrences. Then, SSFIM-EL performs a loop over each transaction. For the transaction $T_i$ of the $i^{th}$ database, SSFIM-EL computes the set of itemsets $S$ for that transaction. For example, if a transaction $T_j$ contains the items $a$, $b$, and $c$, then $S$ will contain the itemsets $a$, $b$, $c$, $ab$, $ac$, $bc$, and $abc$.

Afterwards, SSFIM-EL updates the hash table with the information regarding each new generated itemset $t$ in $S$. If $t$ already exists as a key in $h$ then the corresponding value (the occurrence count of $t$) $h(t)$ in $h$ is incremented by one. Otherwise, a new entry with $t$ as key is created in $h$ and the value $h(t)$ is set to one. Finally, for each entry $t$ in $h$, if the support of $t$ exceeds the minimum support threshold $\text{minsup}$, then the itemset is added to the set of frequent itemsets $F$, which is returned to the user. The overall process of SSFIM-EL terminates when all databases are been processed.

5.3. Example of Extracted Patterns

Having presented the method for extracting frequent itemsets from transactional databases obtained from an event log, the following paragraphs describe some example of patterns that can be extracted thanks to the two pre-processing strategies: creating categorical attributes and creating multiple transactional databases.

Patterns Extracted using Categorical Attributes. Consider the event log of Table 5, where the two categorical attributes Quarter and Duration have been created...
Algorithm 1 The SSFIM-EL Algorithm

1: **Input:** $T = \{T^1, T^2, ..., T^d\}$: $d$ transactional databases. $\minsup$: a user-defined minimum support threshold.
2: **Output:** $F$: The set of frequent itemsets.
3: $F \leftarrow \emptyset$.
4: for each transactional database $T^i$ do
5:     Initialize a hash table $h$.
6:     for each transaction $T^i_j$ do
7:         $S \leftarrow \text{GenerateAllItemsets}(T^i_j)$.
8:         for each itemset $t \in S$ do
9:             if $t \in h$ then
10:                $h(t) \leftarrow h(t) + 1$.
11:            else
12:                $h(t) \leftarrow 1$.
13:            end if
14:         end for
15:     end for
16:     for each itemset $t \in h$ do
17:         if $h(t) \geq \minsup$ then
18:             $F \leftarrow F \cup \{t\}$.
19:     end if
20: end for
21: end for
22: return $F$. 

Table 5: A log event database

<table>
<thead>
<tr>
<th>ID</th>
<th>Activ.</th>
<th>Resour.</th>
<th>Quart.</th>
<th>Dur.</th>
<th>PA</th>
<th>FA</th>
<th>PR</th>
<th>FR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Pete</td>
<td>1</td>
<td>long</td>
<td>?</td>
<td>B</td>
<td>?</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>John</td>
<td>2</td>
<td>short</td>
<td>A</td>
<td>A</td>
<td>Pete</td>
<td>John</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>John</td>
<td>1</td>
<td>short</td>
<td>B</td>
<td>A</td>
<td>John</td>
<td>Lycia</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>Lycia</td>
<td>1</td>
<td>long</td>
<td>B</td>
<td>?</td>
<td>John</td>
<td>?</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>Katia</td>
<td>3</td>
<td>medium</td>
<td>A</td>
<td>?</td>
<td>John</td>
<td>?</td>
</tr>
</tbody>
</table>

using the approach previously described. In this table the following abbreviations are used: Actv.: Activity, Resour.: Resources, Quart.: Quarter, Dur.: Duration, PA: Preceding Activity, FA: Following Activity, PR: Preceding Resource, and FR: Following Resource. If the minimum support threshold on each transactional database is set to 30 \( (\text{minsup} = 30\%) \), several insightful frequent itemsets are extracted, including the following:

- \( F_1: (\text{Duration} = \text{short}, \text{Supp} = 40\%) \) indicates that 40\% of the activities had a short duration.
- \( F_2: (\text{Quarter} = 1, \text{Supp} = 60\%) \) means that 60\% of the activities were performed during the first quarter of the year.
- \( F_3: (\text{Activity} = \text{short PA} = B, \text{Supp} = 40\%) \) means that the activity A is often preceded by activity B, which a support of 40\%.
- \( F_4: (\text{Resources} = \text{John Duration} = \text{short}, \text{Supp} = 40\%) \) means that 40\% of the events have been initiated by John and had a short duration.

**Pattern Extracted from Multiple Transactional Databases.** Some patterns are next presented that can be extracted from the event log of Table 1 after its transformation into ABS-databases. Consider the transformed databases \( T^{(1)} \) and \( T^{(2)} \), which have been previously described, and that \( \text{minsup} = 30\% \). Several frequent itemsets are extracted from these databases, including:

- \( F_1: (\text{Resource} = \text{Mike}, \text{Supp} = 60\%) \) indicates that 60\% of the activities involved a person called Mike.
- \( F_2: (\text{Activity} = \text{RegisterRequest}, \text{Supp} = 60\%) \) means that 60\% of the activities are register requests.
- \( F_3: (\text{Type} = \text{Completed}, \text{Supp} = 60\%) \) and \( F_4: (\text{Activity} = \text{RegisterRequest}, \text{Supp} = 40\%) \) with \( F_3 \) and \( F_4 \), indicates that register request activities have frequently been completed \( (\frac{40}{60} = 66\%) \).
- \( F_5: (\text{tr}_2^{(1)}, \text{Supp} = 66\%) \) means that 66\% of activities are included in both traces \( \text{tr}_1 \) and \( \text{tr}_2 \).
- \( F_6: (\text{tr}_3^{(1)}, \text{Supp} = 66\%) \) means that 66\% of activities appear in the trace \( \text{tr}_3 \).
- \( F_7: (\text{tr}_1^{(2)}, \text{Supp} = 100\%) \) means that the first trace uses all resources.
• $F_8:((\text{tr}_1,\text{tr}_3)^{(2)}, \text{Supp} = 66\%)$ means that the traces $\text{tr}_1$ and $\text{tr}_3$ share 66% of resources.

This subsection has shown that the proposed AllMining method can discover different kinds of insightful patterns from an event log. The next section explains how post-processing can be applied to these patterns to obtain a smaller set of interesting patterns.

6. Step 3: Process Knowledge Discovery

A drawback of frequent itemset mining is that it can generate a high number of frequent itemsets. This is especially true for databases generated from event logs using the proposed approach since adding categorical attributes increase the number of items, and database transformations to events-based and attributes-based databases increase database size. Analyzing a large number of itemsets is time-consuming and inconvenient for the user. To address this issue, this paper proposes a third step for analyzing process models, which consists of filtering the frequent itemsets found in the second step. The purpose is to show to the user a small set of representative itemsets that well describes the event log. A pruning strategy is designed based on a novel concept of coverage to only keep the smallest itemsets that cover the largest number of events from an event log. It is assumed that these itemsets are more useful to the user, as per the Minimum Description Length principle [37], since these patterns are the smallest ones that explain the largest amount of data. Applying the proposed strategy can greatly reduce the number of frequent itemsets, as it will be shown in the experimental evaluation.

It is to be noted that the proposal made in this paper is different from other studies on frequent itemset mining that reduce the number of itemsets by discovering itemsets that are maximal [38] and closed [39]. The key difference is that these studies focus on finding itemsets that are respectively maximal with respect to a database or sets of transactions, rather than finding minimal itemsets. The proposal is also different from studies on generator itemsets as these studies find minimal itemsets with respect to sets of transactions that do not necessarily cover many events from an event log [40]. The next paragraph describes the proposed approach.

Let $F = \{F_1, F_2, ..., F_r\}$ be the set of all frequent itemsets found by the second step. The goal of the third step is to find a subset $F' \subset F$ such that:

$$Pruning_{\text{max}} : F \rightarrow \mathbb{R}$$

$$F' \mapsto Pruning_{\text{max}}(F')$$

where $Pruning_{\text{max}}$ is the itemset pruning function to be maximized. The proposed pruning function is called the coverage pruning function. It is defined as finding the frequent itemsets that cover the maximum number of events from the events of the event log. Formally, let $E = \{e_1, e_2, ..., e_m\}$ be the set of all events and let the notation $Events(F_i)$ denote the set of events covered by an itemset $F_i$. The coverage pruning function is defined as:

$$Pruning_{\text{max}} : F \rightarrow \mathbb{R}^+$$

$$F' \mapsto |\bigcup_{F_i \in F'} Events(F_i)|$$
The optimal solution to this pruning problem is to find the minimal subset $F^*$, where

$$
\begin{align*}
Pruning_{\text{max}}(F^*) &= m \\
\forall F' \subseteq F, \\
Pruning_{\text{max}}(F') &= Pruning_{\text{max}}(F^*) \Rightarrow |F'| \geq |F^*|
\end{align*}
$$

Finding an optimal subset that satisfy the constraints of the coverage pruning function is an NP-complete problem, since the set of frequent itemsets $F$, there are $2^r$ possible subsets of $F$ to choose from. Hence, an exhaustive search would be very time-consuming or even impractical if the cardinality of $F$ is large. To deal with this issue, a greedy search method is developed to reduce the search space and find a good solution rather than a globally optimal solution. The pseudo-code of the search method is shown in Algorithm 2. It takes as input the set of frequent itemsets $F$, a maximum number of iterations $IMAX$ and the number of events in the event log. The output is a set of itemsets $F^*$. The search procedure first creates an initial solution $S$ by randomly selecting a frequent itemset from $F$. Then it puts this solution in a variable $F^*$ representing the current best solution. An iterative process is carried out to improve the current solution to try to find a better solution. This process is repeated until the number of events covered by the itemsets in $F^*$ is less than $m$ or the number of iterations is less than the maximum number of iterations. To improve a solution $S$, the neighborhood $\text{neighbors}$ of that solution is calculated. This consists of creating all solutions that can be obtained by adding another frequent itemset to the current solution $S$. Then, if the best solution among these solutions, denoted as $\text{best}$, is better than the current best solution $F^*$ according to the pruning function, the variable $F^*$ is set to $\text{best}$. The evaluation of the given solution $s$ is computed by using the $Pruning_{\text{max}}$ function, which aims at maximizing the number of events covered by $s$. For each pass of the algorithm, the solution $\text{best}$ maximizes the $Pruning_{\text{max}}$ function. Note that in the above process, if two solutions $S_1$ and $S_2$ are compared such that $Pruning_{\text{max}}(S_1) \leq Pruning_{\text{max}}(S_2)$ and $|S_1| \leq |S_2|$, then the solution $S_1$ is preferred to $S_2$, as we want to minimize the number of itemsets.

For example, consider the following event log depicted in Table 7, which contains 8 events. The subset $F'_1 = \{A, Pete\}$ covers the events $\{e_1, e_2, e_4, e_6, e_7, e_8\}$, which means 75% of the events, while $F'_2 = \{(A, short), (B, John)\}$ covers the events $\{e_1, e_2, e_3, e_5, e_7\}$ that represents only 62% of events. Thus, the solution $F'_1$ would be preferred to $F'_2$.

The time complexity of this step is in the worst case $O(|F| \times |\text{neighbors}| \times IMAX)$, where $F$ is the set of all frequent itemsets, $\text{neighbors}$ is the set of neighbors of the given solution, and $IMAX$ is the maximum number of iterations.

In this section, an approximate incomplete greedy algorithm has been presented to select a set of frequent itemsets that is small and maximize the number of events covered by these itemsets. It is to be noted that other pruning functions could be used for other needs.
Algorithm 2 Pruning Algorithm

1. **Input**: $F = \{F_1, F_2, ..., F_r\}$: the set of all frequent itemsets. IMAX: maximum number of iterations. m: the number of events.
2. **Output**: $F^*$: The set of optimal frequent itemsets.
3. $S \leftarrow $ InitialSol(F).
4. $F^* \leftarrow S$.
5. iter $\leftarrow 0$.
6. **while** $Pruning_{max}(S) < m$ and iter $< \text{IMAX}$ **do**
7. neighbors $\leftarrow \text{ComputeNeighbors}(S)$.
8. best $\leftarrow \text{BestNeighbors}(\text{neighbors})$.
9. **if** $Pruning_{max}(\text{best}) > F^*$ **then**
10. $F^* \leftarrow \text{best}$.
11. **end if**
12. iter $\leftarrow \text{iter} + 1$.
13. **end while**
14. **return** $F^*$

<table>
<thead>
<tr>
<th>Events</th>
<th>Activity</th>
<th>Resource</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>A</td>
<td>Mike</td>
<td>short</td>
</tr>
<tr>
<td>$e_2$</td>
<td>A</td>
<td>Pete</td>
<td>short</td>
</tr>
<tr>
<td>$e_3$</td>
<td>B</td>
<td>John</td>
<td>long</td>
</tr>
<tr>
<td>$e_4$</td>
<td>A</td>
<td>John</td>
<td>medium</td>
</tr>
<tr>
<td>$e_5$</td>
<td>B</td>
<td>John</td>
<td>short</td>
</tr>
<tr>
<td>$e_6$</td>
<td>A</td>
<td>John</td>
<td>long</td>
</tr>
<tr>
<td>$e_7$</td>
<td>A</td>
<td>Pete</td>
<td>short</td>
</tr>
<tr>
<td>$e_8$</td>
<td>A</td>
<td>John</td>
<td>long</td>
</tr>
</tbody>
</table>

Table 6: event log Example
7. Evaluation

Extensive experiments have been carried out to evaluate the performance of the proposed \textbf{AllMining} approach, and in particular its three steps (preprocessing, frequent itemset mining and process knowledge discovery). In a first experiment, the performance of AllMining is compared with that of an improved version of Episode-mining [25] (which employs a frequent itemset mining approach and a pruning heuristic to discover process models) in terms of runtime, memory consumption and number of frequent itemsets found. The reasons for comparing the proposed approach with Episode-mining is that both utilize frequent itemset mining, although in different ways. Then, a second experiments compares the runtime performance and quality of the discovered models obtained by the designed AllMining approach and several state-of-the-art approaches for process model discovery, namely $\alpha$ algorithm [4], ImprovedInductive-miner [11], Declare-miner [8] and Episode-miner [25]. Finally, the knowledge extracted by AllMining is discussed in more details for two case studies of complex processes (Lasagna and Spaghetti processes). All algorithms have been implemented in Java and experiments have been run on a desktop machine equipped with an Intel Core i3 processor and 4 GB of RAM memory.

Runtime and memory consumption measurements were collected using the standard Java memory API. The quality of discovered models was assessed using the following fitness function [11]:

\[
\text{Fitness}(E, M) = \frac{|\{e \in E \land e \models M\}|}{|E|}
\]

where $E$ is an event log instance, $M$ is the model discovered by a given algorithm from that event log, and $e \models M$ is the set of events respecting the model $M$.

In the experiments, the following event log instances were used:

- **Medium and large instances**: Four medium instances have been used, which are a collection of artificial event logs describing four variants of a simple loan application process. \textit{Variant 1} is the most complex process, which includes parallelism and choices. \textit{Variant 2}, 3 and 4 have a simpler and more sequential, control flow, and some activities of \textit{Variant 1} are missing or are split into pairs of activities. These event logs have been widely used to test different approaches for discovering configurable process models from a collection of event logs. Each of these instances contains 2440 events and 475 traces. These instances can be downloaded from \url{https://data.4tu.nl/repository/}.

  A fifth instance used in the experiments is a large instance called \textit{Purchasing instance}, which contains 9119 events and 459 traces. It is obtained from \url{http://www.fluxicon.com/}.

- **Big instances**: Two big instances have been used. The first one, called \textit{Road Traffic Fine Management Process}, was created by de Leoni, M. and Mannhardt, F. It is a real-life event log of an information system managing road traffic fines,
containing 561,470 events and 150,370 traces. This instance is a Lasagna process. It is simple to analyze this instance and extract information from the activity graph because traces are sparse and relatively short (they do not contain too many events per trace). The second big instance, called Hospital, was created by van Dongen, B.F on 2011. It is a real life event log of a Dutch academic hospital, originally intended for use in the first Business Process Intelligence Contest (BPIC 2011). It contains 150,029 events and 1,143 of traces. This instance is modelled as a Spaghetti process. It is difficult to analyze and extract information from the activity graph of that instance because traces are dense and contain a high number of events per trace. Both instances are published by the Eindhoven University of Technology and can be downloaded from https://data.4tu.nl/repository/collection:event_logs_real

7.1. Scalability of AllMining and Episode-mining

In the first experiment, we compared the scalability of the designed AllMining approach with an improved version of Episode-mining [25]. Episode-mining was chosen for this comparison since it is the most recent process mining technique that relies on frequent itemset mining for analyzing event logs. Algorithms were run on the various datasets (instances), and the execution time, maximum memory usage and number of frequent itemsets were compared for different minsup values. The results are shown in Figures 2, 3 and 4, respectively.

A first observation is that when the minimum support is decreased from 100% to 0%, for the medium and large event log instances (labeled (1), (2), (3), (4) and (5)), both algorithms do not exceed 40 seconds, consume less than 1800 MB of memory, and find no more than 350 frequent itemsets.

Moreover, it can be observed that although there is a small gap between the performance of AllMining and Episode-mining for all event logs, AllMining performs best. For the big event log instances (labeled (6) and (7)), when the minimum support is decreased from 100% to 0%, both algorithms spend more than 9000 seconds, consume more than 3600 MB of memory, and find no more than 3500 frequent itemsets. For the big event log instances, AllMining always outperforms Episode-mining, and the gap in terms of performance is large.

Overall, AllMining outperforms Episode-mining on all datasets. It is faster, consumes less memory, and extracts less patterns than Episode-mining.

Furthermore, as the algorithms are applied on event logs containing more events, the performance gap between the two algorithms increases. The reasons for the good performance of the proposed AllMining algorithm are:

1. The small runtime and low memory consumption of AllMining is achieved thanks to its use of the modified SSFIM algorithm. This latter is applied by performing a single scan of the EBS and ABS databases. On the other hand, Episode-mining utilizes an approach inspired by the classical Apriori algorithm to extract frequent itemsets. This approach requires to perform multiple database scans, which are very costly for large and big instances.
Figure 2: CPU Runtime comparison of AllMining and Episode-mining for: Loan Application Process.variant1 (1), Loan Application Process.variant2 (2), Loan Application Process.variant3 (3), Loan Application Process.variant4 (4), Purchasing Instance (5), Road Traffic Fine Instance (6), and Hospital Instance (7).
Figure 3: Memory Consumption (mb) of AllMining and Episode-mining for: Loan Application Process.variant1 (1), Loan Application Process.variant2 (2), Loan Application Process.variant3 (3), Loan Application Process.variant4 (4), Purchasing Instance (5), Road Traffic Fine Instance (6), and Hospital Instance (7)
Figure 4: The number of frequent itemsets returned by AllMining and Episode-mining for: Loan Application Process.variant1 (1), Loan Application Process.variant2 (2), Loan Application Process.variant3 (3), Loan Application Process.variant4 (4), Purchasing Instance (5), Road Traffic Fine Instance (6), and Hospital Instance (7)
2. The number of itemsets found by AllMining is smaller than that of Episode-Mining due to the proposed coverage pruning strategy, which finds a small subset of all frequent itemsets that covers a maximum number of events from an event log.

Although AllMining outperforms Episode-mining, it does not mean that AllMining extracts the best process models. Thus, the next paragraphs compare the quality of the models extracted by AllMining with those extracted by Episode-Mining and several other state-of-the-art process model mining algorithms.

7.2. AllMining Vs state-of-the-art process model mining algorithms

This subsection presents a comparison of AllMining with the state-of-the-art process model mining algorithms Inductive-Miner [11], Declare-Miner [8] and α-Miner [4]. For this experiment, the same seven datasets have been used. The percentage of the dataset size has been varied from 0% to 100%, while measuring the execution time and the quality of the models discovered using Equation 1 for each algorithm. Results are shown in Figures 5 and 6.

In terms of execution time, results indicates that α-Miner outperforms the other algorithms for all datasets. Moreover, it can be observed that AllMining is very competitive. It outperforms Inductive-Miner and Declare-Miner, and its performance is very close to that of α-Miner.

In terms of quality of the discovered models, AllMining provides much better models compared to the other algorithms. It reaches an optimal fitness value of 1.0 for almost all event log instances, regardless of the dataset size used. Moreover, although α-Miner is faster, it is the worst in terms of model quality.

The reasons why the proposed AllMining approach performs well in this experiment are:

1. AllMining is faster than Inductive-Miner and Declare-Miner thanks to its adapted SSFIM algorithm, which avoids performing multiple database scans.
2. AllMining outperforms the state-of-the-art process model mining algorithms in terms of model quality thanks to its more detailed methodology involving three steps: preprocessing, extracting frequent itemsets, and filtering these frequent itemsets to extract a concise and informative process model. The knowledge extracted by AllMining is rich since AllMining consider multiple databases (EBS and ABS databases) representing multiple and varied perspectives on the data. Moreover, the pruning function proposed in this work allows to build a small process model that is representative of a maximum number of events. In contrast, the other approaches attempt to discover process models by only using the initial event log (they do not perform database transformations). Thus, these approaches ignore several important patterns that could be found by transforming the data.
3. AllMining achieves higher fitness than that of α-Miner, while the latter is faster than AllMining. This is because the steps performed by AllMining are more complex than those performed by α-Miner. Thus, AllMining extract better models than α-Miner but requires more time.
Figure 5: Runtime comparison of AllMining and state-of-the-art process model mining algorithms for: Loan Application Process.variant1 (1), Loan Application Process.variant2 (2), Loan Application Process.variant3 (3), Loan Application Process.variant4 (4), Purchasing Instance (5), Road Traffic Fine Instance (6), and Hospital Instance (7)
Figure 6: Fitness comparison of AllMining and state-of-the-art process model mining algorithms for: Loan Application Process.variant1 (1), Loan Application Process.variant2 (2), Loan Application Process.variant3 (3), Loan Application Process.variant4 (4), Purchasing Instance (5), Road Traffic Fine Instance (6), and Hospital Instance (7)
Table 7: Number of Frequent Itemsets Before/After Pruning for Road Fine Process (Lasagna Case) and Hospital (Spaghetti Case)

<table>
<thead>
<tr>
<th>Instances</th>
<th>Before Pruning</th>
<th>After Pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoadFineProcess</td>
<td>715</td>
<td>19</td>
</tr>
<tr>
<td>Hospital</td>
<td>614</td>
<td>33</td>
</tr>
</tbody>
</table>
Figure 9: The set of Frequent Itemsets that covers all the event log for Road Fine Process (Lasagna Case) and Hospital (Spaghetti Case)
7.3. Case Studies

Having compared the performance of AllMining with other approaches in previous subsections, this subsection focus on the AllMining approach by presenting two detailed case studies illustrating the patterns found by AllMining. The two case studies are big event log instances. The first one is the Road Fine Process instance which is a lasagna process, whereas the second one is the Hospital instance, which is a spaghetti process. The next paragraphs first illustrates the patterns found by adding categorical attributes to an event log, then the patterns found thanks to the construction of multiple databases, and finally the patterns found by applying the coverage pruning function.

1. Adding Categorical Attributes

Figure 7 shows relevant frequent itemsets obtained by adding additional information such as the time of the year and duration to the Road Fine Process and Hospital instances. Results show that interesting knowledge can be extracted using the proposed approach such as:

- (Complete Fourth Quarter, support=25%): 25% of the activities have been completed during the fourth quarter of the year.
- (General Lab Clinical Chemistry First Quarter, support=33%): 33% of the operations have been performed during the first quarter of the year by General Lab Clinical Chemistry.
- (First Quarter, support=39): 39% of activities have started during the first quarter of the year.

2. Constructing Multiple Transactional Databases

Figure 8 shows the relevant frequent itemsets obtained by constructing multiple transactional databases for the Road Fine Process (Lasagna Case) and the Hospital (Spaghetti Case) event logs. The results shows that interesting and novel knowledge are extracted such as:

- (Tr_{Ressources}5, support=25%): 25% of doctors participates in the 59th trace.
- (Tr_{Type}1 Tr_{Type}2, support=28%): 28% of the activities are of the same type in the first and the second trace
- (Tr_{Activity}1 Tr_{Activity}2, support=31%): The first and the second trace share 31% of the activities.

3. Coverage Pruning Function

Table 7 presents the number of frequent itemsets obtained before and after applying the coverage pruning function on the two instances (Road Fine Process and Hospital). This table shows that the number of frequent itemsets presented to the user is greatly reduced by applying the pruning step. Only 19 itemsets are produced instead of 715 for Road fine Process, and only 33 instead of 614 for Hospital. Note that the optimal subsets of frequent itemsets found by AllMining perfectly describe the information derived from the event log, as it covers all events.

Figure 9 depicts the optimal subset of frequent itemsets extracted for Road fine Process and Hospital.
7.4. Discussion

For the sake of conciseness, the remainder of this section discusses the main findings derived from the application of the proposed approach to real event logs without providing the complete list of frequent itemset extracted for each case and strategy.

The first finding of our evaluation is that pruning using the coverage pruning function aims to find interesting knowledge. The purpose of pruning is to select frequent itemsets based on the number of items (attributes) that they cover. This gives some meaningful results that can support process analysis. Frequent itemsets covering a small number of items (attributes) provides information about general patterns in the event log, whereas frequent itemsets involving many items (attributes) represent specific patterns in an event log. General patterns reveal information about frequently executed activities, busy periods of the year, i.e., in which many activities have been completed, and busy resources, i.e., resources involved in many activities. For instance, in EL1, 1-attribute itemsets are First Quarter, Second Quarter, 541 (resource id), which signifies that most events in this event log have been recorded in the first half of the year and that resource 541 has been involved in a high number of events. In EL2 (Hospital), 1-attribute itemsets are mainly the most frequent activities and wards involved in events. More specific knowledge patterns are captured by 2-attributes and 3-attribute itemsets.

Since the proposed approach relies on a data mining technique, it has an inductive and predictive character. In the context of process mining, we argue that current tools are generally suitable for ex-post analysis of business process properties, but lack an inductive character. Current tools enable the quick identification of knowledge involving one attribute, e.g., time-related or resource-related values. Although current tools can also be used to verify frequent patterns correlating multiple attributes, the identification of these patterns requires specialist skills and intuition of a business analyst. For example, it would be easy to confirm that a correlation holds using a process mining tool such as ProM. However, identifying this correlation would require specialist skills by a business analyst. It requires to find and interpret information from event logs that can be very large. On the contrary, the proposed approach provides a quick glance at relevant patterns in an event log, and this process is automatic.

In this context, we argue that our approach is aligned with an emerging trend in business intelligence that shifts the intelligence required for identifying virtuous or vicious patterns from the tacit knowledge of the business analyst to the data analysis tool, which proactively should suggest areas of interest for further investigation.

From a data mining research standpoint, our paper is an example of the application of a generic data mining technique to a specific context. The literature calls for this type of research, particularly in the times of Big Data where increasingly large amounts of data become available in different domains. As in many other cases, porting a pure data mining technique into a specific application domain requires methodological refinement and adaptation. In our specific context, this adaptation is implemented in different phases such as adding additional attributes, generating multiple transactional databases from an event log and applying pruning to reduce the number of frequent itemsets to a number that can be handled by decision makers.
8. Conclusions

This paper presented an approach named AllMining to extract knowledge about business processes from event logs using frequent itemset mining. Applying frequent itemset mining to event logs requires several methodological adaptations. The proposed approach thus consists of three steps. The first step consists of preparing an event log for data analysis. Two transformations are proposed. The first one is to create categorical attributes based on existing attributes to support specific knowledge extraction objectives. The second one is to create ABS and EBS databases to analyze an event log with respect to its attributes and events, to provide multiple perspectives on the data. The second step is to extract frequent itemsets. For this purpose, any frequent itemset mining algorithms could be used. In this work, the SSFIM algorithm has been adapted to reduce the number of database scans. The third step then consists of filtering the frequent itemsets found to show a reasonable number of relevant frequent itemsets to decision makers. For this purpose, a coverage pruning function has been presented that selects a small groups of itemsets that covers the maximum number of events. Our approach has been tested analytically to analyze its performance along several dimensions, and has been evaluated with real event logs. The evaluation shows that our approach enables the pro-active discovery of knowledge in an event log, which can trigger further investigation, such identifying best practices or process redesign.

Future work should look along different directions. First, we plan to work on embedding the core approach described in this paper in a decision support system for business process analysts, which could be tested with practitioners. This would require efforts particularly in the design of the human interface to the frequent itemset, which should allow for an efficient presentation of the results and an efficient navigation among discovered itemsets. We are also working on refining the technical core of our approach. In particular, we are developing domain-specific pruning and organization techniques that could take into account specific properties of the process under observation or specific interests of the decision maker.

References


[28] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in the proceeding of ACM Sigmod Record. 29(2) (2000) 1-12.


