Relational Disjunctive Patterns Mining for Discovering Frequent Variants in Process Models

Corrado Loglisci, Michelangelo Ceci, Annalisa Appice, and Donato Malerba

Dipartimento di Informatica, Università degli Studi di Bari
via Orabona, 4 - 70126 Bari - Italy
{loglisci, ceci, appice, malerba}@di.uniba.it

Abstract. The automatic discovery of process models can help to gain insight into various perspectives (e.g., control flow or data perspective) of the process executions traced in an event log. Frequent patterns mining offers a means to build human understandable representations of these process models. This paper describes the application of a multi-relational method of frequent pattern discovery into process mining. Multi-relational data mining is demanded for the variety of activities and actors involved in the process executions traced in an event log which leads to a relational (or structural) representation of the process executions. Peculiarity of this work is in the integration of disjunctive forms into relational patterns discovered from event logs. The introduction of disjunctive forms enables relational patterns to express frequent variants of process models. The effectiveness of using relational patterns with disjunctions to describe process models with variants is assessed on real logs of process executions.

1 Introduction

Workflow management systems are becoming increasingly important in enterprises due to their capabilities of managing activities and actors involved in a business process as well as recording the logs of process executions.

Despite the amount of event logs produced by enterprises, software vendors use this information in order to answer to only simple questions under the assumption that the business process is fixed and known, e.g., the calculation of performance metrics like utilization and flow time. However, in many domains, business processes are evolving and people may have an oversimplified and incorrect view of the actual business processes [10]. In this scenario, process mining techniques play a key role with the extraction of models (or patterns) from event logs where they can provide useful insights in the design of new workflows as well as they permit to collect information exploitable in the workflow optimization.

In the literature, several approaches for mining process models have been proposed. A Markovian approach is described in [2] to investigate the correspondence between the instances of a software engineering process and a model of the process. The process model is represented by a Finite State Machine derived from executions of software development processes and, with the help of

distance metrics, it is used to quantitatively measure the discrepancies from new executions. A graph-based perspective is considered in [12] where a variant of Petri nets, called Workflow Nets, has been used to mine and model workflow processes. In these works, a sophisticated algorithm is presented to extract a process model based on binary relations discovered into the workflow logs. Notably, this approach permits to determine which class of workflow models the algorithm is guaranteed to work with. More recently, some authors [5], as we do in this work, have successfully applied pattern discovery techniques in order to identify frequent activities and their relationships. Since high frequency denotes regularity, frequent patterns can provide arguments for process models based on the evidence of regularities in the executions. Indeed, frequent patterns are intended as a means to capture the typical order of execution between activities (control perspective) and, at the same time, they model the possible associations among the properties of the process, activities and actors (data perspective).

The common characteristic of studies reported above is that they discover patterns (or more generally models) that identify the typical order of execution of activities without considering *variants*. However, real-world processes tend to be so complex and less structured that it is difficult to determine patterns to which several process executions comply with [11]. Moreover, the application of patterns that does not consider variants can turn out to be impractical in the workflow management systems which deal with exceptional situations and structural changes during runtime. On the other hand, considering these changes when discovering patterns may lead to the generation of numerous variants which are difficult to maintain even if slightly different one from each other [4, 7].

A deep analysis may reveal that this limitation comes from the fact that traditional frequent pattern discovery algorithms permit to mine conjunctions of the activities present in a set of process executions and does consider disjunctions that can model process variants. This approach poses some limitations to the patterns expressiveness and, in addition, leaves unexplored two potentialities of the pattern discovery: i) discovering interesting patterns when activities are not present in a sufficient number of process executions, and ii) discovering a combination of relationships between activities which are different from the classical conjunctions, such as disjunctions. These two potentialities are not independent each other, since the discovery of patterns including other relationships between activities may lead to discover patterns that otherwise would be discarded. Considering relationships among activities different from the classical conjunction would permit not only to consider variants in the process executions, but also to consider parallel executions of activities.

In this work, we extend our work in [1], where we have investigated the discovery of frequent patters as a means to extract a human interpretable representation of process models. The peculiarity of our previous work is that the frequent pattern discovery is performed in multi-relational data mining in order to take into account the intrinsic relational structure of logs: several activities and/or actors are involved in the same process execution. In particular, the multi-relational approach permits to solve the following problems. First, objects

collected in a log belong to different data types (executions, activities and actors) which interact one each other. By resorting to the first-order logic, that is one of the most common relational representation formalism, properties and interaction of executions, activities and actors are modeled by means of first-order logic predicates. Then, reasoning techniques developed in the field of inductive logic programming (ILP) are employed to discover patterns which are relationally defined as conjunctions of atomic formulas built using the data predicates. Second, activities stored in a log are marked with a timestamp which indicates the time of occurrence and implicitly defines a total temporal order over events. Temporal relationships between activities are represented as predicates and these predicates are used to investigate the temporal autocorrelation in the effect of a property of an activity/actor. Third, some user defined domain knowledge (e.g., the definition of the ordering relation between activities) may be available. This knowledge is profitably exploited by inferential mechanisms typical of a theorem prover which are integrated in the relational pattern discovery. However, the main disadvantage of the algorithm presented in [1] is that it cannot deal with process variants that, as stated before, are of fundamental importance.

To cope with this further issue, we propose to derive process models through the discovery of relational patterns with disjunctions. The advantage of integrating disjunctive forms in patterns is two-fold. First, process variants can be identified and represented with the reference model, thus avoiding the explicit maintenance of numerous variants. Second, activities in the patterns can be OR-ed to represent typical $OR-split/OR-join$ in graph-based constructs [12]. The paper is organized as follows. In the next two sections we present related works and background. In Section 3 we present our algorithm to discover disjunctive relational patterns. Experimental results on two real-world databases are commented in Section 4 and conclusions are drawn.

2 Related Work and Background

Recently, the multi-relational or ILP approaches to build business process models from event logs are receiving increasing attention. Goedertier et al. [3] have faced the task of predicting, by means of learned relational classification rules whether, given the state of a process instance, a particular state transition can occur. The representation formalism considered in this work is the Event Calculus, a first-order logic that elegantly captures the time-varying nature of facts. Learning is based on both positive information (possible transitions) and negative information (prohibited transitions). When no negative information is actually available in the logs, it is artificially generated by means of the closed-world assumption.Similarly, Lamma et al. [6] have considered both compliant (positive information) and non compliant (negative information) execution traces and adapt the algorithm ICL to learn constraints among activities expressed as logical formulas. In practice, the main problems of both methods are the reliable provision of negative information and their scalability to huge event logs.

We overcome this issue by learning from positive examples only (as usual in frequent pattern discovery).

Mining variants of a process is not a novel task. Indeed, significant studies have already addressed this task in the literature. Li et al. [7] describe an approach to discover the reference block-structured model which is the best in covering process variants. The model is obtained by first clustering activities in order to form blocks of similar activities, and then merging blocks into larger blocks. The final reference model is thus the model which has minimum distance from the variants, where the distance is measured by the number of change operations at the activity level. The brittleness of this work is the difficulty to create a set of only variants given that, in real event logs, traces of process and variants are stored together. The determination of a set of workflow models, called *disjunctive workflow schema*, is rather the solution proposed by Greco et al [4] to model the relevant variants. A stepwise procedure permits to refine models (workflow schema) created at the previous step, so that the final set is composed of hierarchically organized models. At each step, the executions which support a schema are partitioned into clusters each of which contains executions with the same characteristics. Each cluster thus represents a relevant variant and it is modeled by a refined workflow schema. However, since each variant is mined with a specialized model, the applicability of this method may be compromised in the case a huge number of specialized models is produced.

The background of this work is in [1], where we have used SPADA in order to extract relational frequent patterns from event logs. In SPADA, it is possible to distinguish between reference objects (*ro*) and task-relevant objects (*tro*). The former are data on which patterns are enumerated and contribute to compute the support of a pattern, while the latter contribute to define the former and they can be involved in a pattern. In the logic framework adopted by SPADA, event logs are converted into a deductive database D. Properties and relationships of the process executions (reference objects), activities and actors (task-relevant objects) are represented as ground atoms in the extensional part D_E, while a user defined background knowledge is expressed as a normal logic program which defines the intensional part D_I. An example of ground atoms stored into the extensional database D_E is reported in the followings:
$process(e1)$. $process(e2)$. $activity(e1, a1)$. $activity(e1, a2)$. $activity(e2, a3)$.
$activity(e2, a4)$. $is\ a(a1, workflow)$. $is\ a(a2, complete)$. $is\ a(a3, namemaker)$.
$is\ a(a4, schedule)$. $time(a1, 10)$. $time(a2, 25)$. $time(a3, 22)$. $time(a4, 23)$.
$actor(a1, paul)$. $actor(a2, paul)$. $is\ a(paul, user)$. $actor(a3, paul)$.
$actor(a4, mary)$. $is\ a(mary, admin)$.

These ground atoms describe the process executions $e1$ and $e2$ (reference objects) according to the activities $a1$, $a2$, $a3$, and $a4$ (task-relevant objects) and the actors, $u1$ and $u2$ (task-relevant objects). Differently, an example of a normal logic program stored as intensional database D_I is the following:
$before(A1, A2) \leftarrow activity(C, A1), activity(C, A2), A1 \neq A2, time(A1, T1), time(A2, T2),$
$T1 < T2, not(activity(C, A), A \neq A1, A \neq A2, time(A, T), T1 < T, T < T2)$
This normal logic program defines the temporal relationship *before* and permits to entail the temporal ground atoms $before(a1, a2)$ and $before(a3, a4)$.

The set of ground atoms (extensionally or intensionally) stored in D is partitioned with respect to the process executions into a number of non-intersecting subsets $D[e]$ (units of analysis) each of which includes ground atoms concerning the activities and actors involved in the process execution e. Then, SPADA is able to discover relational frequent patterns across the units of analysis of D which are associated to the process executions. The discovery process is in charge of a levelwise method, that is tailored as a breadth-first search in the lattice of relational patterns spanned by the θ-subsumption generality order ($\succ_\theta$). In this context, the relational patterns are formulated as $process(P), \mu(P)$ $[s]$, where P is a variable to represent a process execution, $process(P)$ is the atom that identifies a process execution P, while $\mu(P)$ is a conjunction of atoms which provides a description of a fragment of the process model underlying the generation of the process execution P, s is the support of P in D. Each atom in $\mu(P)$ represents one of the property of activity, actor or process execution, or relationship between activities, process executions and activities, activities and actors. For example:

$process(P), complete(P, A), schedule(P, B), delete(P, C)$ $before(A, B)$, $before(B, C)$. $[support = 63\%]$

is a relational pattern which describes a fragment of a process model where the activities complete, schedule and delete are executed in a sequence. The support s estimates the probability $p(process(P) \cup \mu(P))$ on D. This means that $s\%$ of the units of analysis $D[e]$ are covered by $process(P) \cup \mu(P)$. Formally, the unit of analysis $D[e]$ is covered by $process(P) \cup \mu(P)$ if there exists a substitution $\theta = \{P \leftarrow e\} \cdot \theta_1$ such that $[process(P) \cup \mu(P)]\theta \subseteq D[e]$.

3 Discovering Process Models with Variants

The motivation behind the usage of disjunctive forms in patterns is that the set of patterns discovered with traditional approaches, included SPADA, strongly depends on frequency-based thresholds such as the minimum support threshold. This means that when the minimum support is high valued, many interesting patterns are filtered out: conjunctions of atoms, for which the considered statistical measure does not exceed the minimum threshold, are ignored. The introduction of the disjunctive forms would permit to include the atoms which occur in parallel to or in alternative to other atoms. The effect is that of increasing the values of the statistical measures associated to the patterns. For example, let us suppose that the atom $complete(A, D)$ may occur alternatively to the atom $schedule(A, D)$. Then, the relational pattern:

$process(A), complete(A, B), \langle \mathbf{complete}(\mathbf{A}, \mathbf{D}) \vee \mathbf{schedule}(\mathbf{A}, \mathbf{D}) \rangle, before(B, D)$

might be frequent, although the patterns

$process(A), complete(A, B)$, $complete(A, D)$, $before(B, D)$ and

$process(A), complete(A, B), schedule(A, D), before(B, D)$

might be both infrequent.

This consideration advocates the starting point of our approach, which is that of considering infrequent conjunctive patterns. These patterns are re-evaluated

and extended to the disjunctive form by inserting disjunctions which involve atoms already present in the patterns. Disjunctions are created among atoms which are semantically related in the application domain. The semantic relatedness is intended as background knowledge on the predicates used in the atoms and permits us to numerically quantify the dissimilarity or conceptual distance between atoms. It guarantees that meaningful disjunctions are created.

The proposed approach follows a two-stepped procedure. First, it extracts the infrequent conjunctive patterns which can be considered as basis for the disjunctive patterns construction. In particular, patterns whose support is lower than the minimum support threshold, but exceeds a new ad-hoc threshold are selected. The newly defined threshold permits to identify the set of patterns to be extended to the disjunctive form. Second, by following the main intuition reported in [9], background knowledge is accommodated to exploit the information on the dissimilarity among the atoms in the disjunctive pattern generation. This way, disjunctive patterns are computed by iteratively integrating disjunctions into the patterns by means of a pair-wise joining operation. The final result is a set of patterns which may comprise both conjunctions and disjunctions of atoms, whose support is greater than the minimum support threshold.

Working in the relational setting adds additional sources of complexity to the problem of joining patterns due to the *linkedness* property [8]. In the relational representation, atoms of the same pattern are dependent each other due to the presence of variables (differently from the items in the propositional representation [9]). In this work, patterns to be joined should differ in only one atom (but different atoms should be similar) and share the remaining atoms up to a redenomination of variables.

Before formally defining the problem we solve in this paper, we clarify how the deductive database we have described in the previous section changes. In particular, the intensional part D_I of the deductive database D includes the definition of a new kind of domain knowledge that permits to express the dissimilarity among atoms in the form of *Datalog* weighted edges of a graph. An example of the Datalog weighted edge is the following: $schedule - (complete - 0.88)$. It states that the dissimilarity between the predicates *schedule(·,·)* and *complete(·,·)* is 0.88. More generally, it represents an undirected edge e between two vertices v_i and v_j (e.g., *schedule*, *complete*) with weight w_{ij} (e.g., 0.88) and it is denoted as $e(v_i, v_j, w)$. A finite sequence of undirected edges $e_1, e_2, \ldots, e_m$ which links the vertices v_i and v_j is called *path* and denoted as $\rho(v_i, v_j)$. The complete list of such undirected edges represents the background information on the dissimilarity among atoms and allows the algorithm to join patterns by introducing disjunctions (e.g., $\langle schedule(A, W) \vee complete(A, W)\rangle$). The formal statement of the problem of discovering relational frequent patterns with disjunctions can be articulated in two steps.

1. *Given:* the extensional part D_E of the deductive database D and the normal logic programs stored into the intensional part D_I of D, two thresholds $minSup \in [0; 1]$ and $nSup \in [0; 1]$, where the former represents a minimum support value, while the latter represents a maximum support value ($nSup <$

minSup), *Find:* the collection I_R of the relational infrequent patterns whose support is between *nSup* and *minSup*.
2. *Given:* the collection I_R, the Datalog weighted edges stored in the intensional part D_I of the deductive database D and two thresholds *minSup* and $\gamma \in [0; 1]$ (γ defines the maximum dissimilarity value among atoms involved into a disjunction), *Find:* relational patterns with disjunctions whose support exceeds *minSup* and whose dissimilarity of atoms involved in the disjunctions does not exceed γ.

The computational solution to these problems is implemented in jSPADA.

3.1 Mining Infrequent Conjunctive Relational Patterns

In jSPADA, the search is based on the level-wise method and implements a two-stepped procedure: i) generation of candidate patterns with k atoms (k-th level) by considering the frequent patterns with $k-1$ atoms ($k-1$-th level); ii) evaluation of the support of patterns with k atoms. So, the patterns whose support does not exceeds *minSup* will be not considered for the next level: the patterns discarded (infrequent) at each level are rather considered for the generation of disjunctions. The collection I_R is thus composed of a subset of infrequent patterns, more precisely those with support greater than or equal to *nSup* (and less than *minSup*).

We observe that, the set of infrequent patterns I_R computed by jSPADA do not contain all possible infrequent relational patterns. This depends on the fact that any k-level infrequent pattern "$P_{k-1} \wedge A$" output by jSPADA has the head P_{k-1} that is a relational pattern frequent at the level $k-1$, and the tail A which is an atom such that the conjunction "$P_{k-1} \wedge A$" is infrequent. This means that infrequent patterns whose head is already infrequent cannot be discovered by jSPADA. On the other hand computing all possible infrequent relational patterns is computationally expensive. As a consequence, the set of patterns with disjunctions discovered from I_R is a necessary approximation of the complete set of relational patterns with disjunctions.

3.2 Extending Relational Patterns with Disjunctions

The generation of disjunctive relational patterns is performed by creating disjunctions among similar atoms in accordance to the weighted edges of the background knowledge: two patterns which present similar atoms are joined to form only one. The implemented algorithm (see Algorithm 1) is composed of two subprocedures: the first one (lines 2-12) creates a graph $\mathcal{G}_\mathcal{D}$ with the patterns of I_R by exploiting the knowledge defined in D_I, while the second one (lines 13-32) joins two patterns (vertices) on the basis of the information (weight) associated to the connecting edge. In particular, for each pair of patterns which have the same length (namely, at the same level of the level-wise search method) it checks whether they differ in only one atom and share the remaining atoms up to a redenomination of variables (line 3). Let α and β be the two atoms differentiating P from Q (α in P, β in Q), a path ρ which links α to β (or vice-versa) is searched

among the weighted edges according to D_I: in the case the sum ω of the weights found in the path is lower than the maximum dissimilarity γ the vertices P and Q are inserted into $\mathcal{G}_\mathcal{D}$ and linked through an edge with weight ω (lines 4-9). Note that when there is more than one path between α and β, then the path with lowest weight is considered. Intuitively, at the end of the first sub-procedure, $\mathcal{G}_\mathcal{D}$ will contain, as vertices, the patterns which meet the condition at the line 3, and it will contain, as edges, the weights associated to the path linking the atoms differentiating the patterns.

Once we have $\mathcal{G}_\mathcal{D}$, a list $\mathcal{L}_\mathcal{D}$ is populated with the vertices and edges of $\mathcal{G}_\mathcal{D}$: an element of $\mathcal{L}_\mathcal{D}$ is a triple $\langle P, Q, \omega \rangle$ composed of a pair of vertices-patterns (P, Q) with their relative weight. Elements in $\mathcal{L}_\mathcal{D}$ are ranked in ascending order with respect to the values of ω so that the pairs of patterns with lower dissimilarity will be joined for first. This guarantees that disjunctions with very similar atoms will be preferred to the others (line 13). For each element of $\mathcal{L}_\mathcal{D}$ whose weight ω is lower than γ the two patterns P and Q are joined to generate a pattern J composed by the conjunction of the same atoms in common to the two patterns P and Q and of the disjunction formed by the two different (but similar) atoms (lines 14-15). This joining procedure permits to have patterns with the same length of the original ones and which occur when at least one of the original patterns occurs. Therefore, if a pattern J is obtained by joining P and Q, it covers a set of units of analysis equal to the union of those of P and Q: the support of J is determined as in line 16 and, generally, it is higher than the support of both P and Q. In the case the support of J exceeds $minSup$, then it can be considered statistically interesting and no further processing is necessary (lines 16-17). Otherwise, J is again considered and inserted into $\mathcal{G}_\mathcal{D}$ as follows. The edges which linked another pattern R of $\mathcal{G}_\mathcal{D}$ to P and Q are modified in order to keep the links from R to J: the weight of the edges between one pattern R and J will be set to the average value of the weights of all the edges which linked R to P and Q (lines 19-27). The modified graph $\mathcal{G}_\mathcal{D}$ contains conjunctive patterns (those of I_R) and patterns with disjunctions (those produced by joining). Thus, $\mathcal{G}_\mathcal{D}$ is re-evaluated to extend disjunctions previously created or to insert into disjunctive patterns other disjunctions obtained with other atoms. The algorithm proceeds iteratively (line 29-30) until no additional disjunction can be performed (namely, when $\mathcal{L}_\mathcal{D}$ is empty or the weights ω are higher than γ). At each iteration, the patterns P and Q are removed from $\mathcal{G}_\mathcal{D}$ (line 32).

An explanatory example is illustrated in Figure 1. Let us consider the background knowledge D_I on the dissimilarity among four atoms and the set I_R containing four infrequent conjunctive relational patterns as illustrated in Figure 1a and γ equal to 0.7. The first sub-procedure of Algorithm 1 analyzes P_1, P_2, P_3, P_4 and discovers that they differ in only one atom, while the other atoms are in common ($process(A), unknown(A, C), before(B, C)$). Then, it creates the graph $\mathcal{G}_\mathcal{D}$ by collocating P_1, P_2 and P_3 in three different vertices and linking them through edges whose weights are taken from the paths ρ in D_I. P_4 is not considered because the dissimilarity between *start* and *resume* in the graph is higher than γ (row (1) in Figure 1b). The second sub-procedure starts

Algorithm 1 Extending Relational Pattern with Disjunctions.

input: $I_R, D_I, \gamma, minSup$
output: $\mathcal{J}$ // $\mathcal{J}$ set of disjunctive patterns
for all $(P, Q) \in I_R \times I_R$, $Q \neq P$ **do**
  if $P.length = Q.length$ and $check\_atoms(P, Q)$ **then**
    $(\alpha, \beta) := atoms\_diff(P, Q)$ //α, β atoms differentiating P and Q
    if $\rho(\alpha, \beta) \neq \emptyset$ **then**
      $\omega := \sum_{e(v_i, v_j, w_{ij}) \ in \ \rho(\alpha,\beta)} w_{ij}$
      if $\omega \leq \gamma$ **then**
        $addNode(P, \mathcal{G}_\mathcal{D})$; $addNode(Q, \mathcal{G}_\mathcal{D})$;$addEdge(P, Q, \omega, \mathcal{G}_\mathcal{D})$;
      end if
    end if
  end if
end for
$\mathcal{L}_\mathcal{D} \leftarrow$ edges of $\mathcal{G}_\mathcal{D}$ // list of edges of $\mathcal{G}_\mathcal{D}$ ordered in ascending mode w.r.t. ω
while $\mathcal{L}_\mathcal{D} \neq \emptyset$ and $\forall e(P, Q, \omega) \in \mathcal{G}_\mathcal{D}$ $\omega \leq \gamma$ **do**
  $J \leftarrow join(P, Q)$; $J.support := P.support + Q.support - (P \cap Q).support$;
  if $J.support \geq minSup$ **then**
    $\mathcal{J} := \mathcal{J} \cup \{J\}$
  else
    for all R *such that* $\exists\ e(P, R, \omega_1) \in \mathcal{G}_\mathcal{D}$ and $\exists\ e(Q, R, \omega_2) \in \mathcal{G}_\mathcal{D}$ **do**
      $addEdge(R, J, (\omega_1 + \omega_2)/2, \mathcal{G}_\mathcal{D})$
    end for
    for all R *such that* $\exists\ e(P, R, \omega_1) \in \mathcal{G}_\mathcal{D}$ and $\nexists\ e(Q, R, \omega_2) \in \mathcal{G}_\mathcal{D}$ **do**
      $addEdge(R, J, \omega_1, \mathcal{G}_\mathcal{D})$
    end for
    for all R *such that* $\exists\ e(Q, R, \omega_2) \in \mathcal{G}_\mathcal{D}$ and $\nexists\ e(P, R, \omega_1) \in \mathcal{G}_\mathcal{D}$ **do**
      $addEdge(R, J, \omega_2, \mathcal{G}_\mathcal{D})$
    end for
    $\mathcal{L}_\mathcal{D} \leftarrow$ edges of $\mathcal{G}_\mathcal{D}$; *update* $\mathcal{L}_\mathcal{D}$
  end if
  $removeNode(P, \mathcal{G}_\mathcal{D})$; $removeNode(Q, \mathcal{G}_\mathcal{D})$
end while

by ordering the weights of the edges: the first disjunction is created by joining P_1 and P_3 given that the dissimilarity value is lower than γ and the lowest (row (2) in Figure 1b). Next, the pattern so created and P_2 are checked for joining. Both have the same length and differ in only one atom. Although the first presents a disjunction and the second presents a "simple" atom, dissimilarity is lower than γ and a new disjunctive pattern is created (row (3) in Figure 1b).

A final consideration concerns the time complexity of Algorithm 1. Let us consider the set I_R partitioned into disjoint subsets $\{I_{Rk}\}_k$ on the basis of the pattern length k. Let n_k be the cardinality of I_{Rk}. For each k, the time complexity of extending relational patterns in I_{Rk} with disjunctions is quadratic in n_k at worst (this is due to the pairwise join operation).

4 Experiments

Experiments are performed by processing event log provided by THINK3 Inc[1]. This dataset describes 353,490 executions of business processes in a company. The period under analysis is from April 7th 2005 to January 10th 2007 for a total of 1,035,119 activities and 103 actors. Activities are classified as *tools* (131),

[1] http://www.think3.com/en/default.aspx

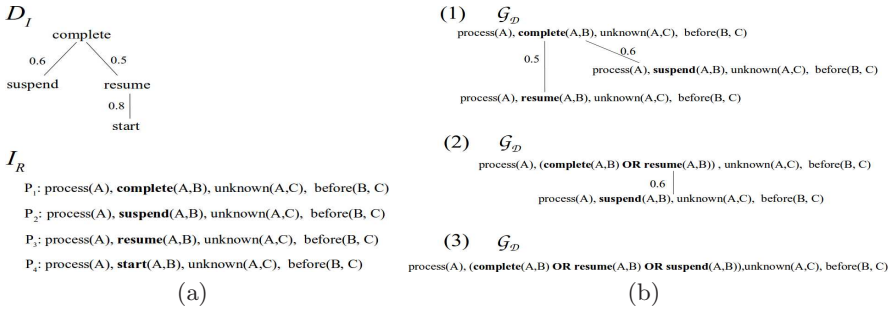

Fig. 1. An example of relational pattern extension from disjunctive atoms (γ=0.7).

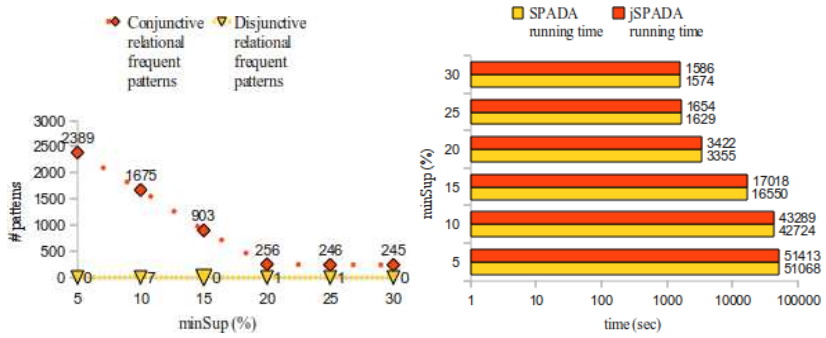

Fig. 2. Learning times and number of patterns discovered by SPADA and jSPADA by varying *minSup* (*nSup*=0.1%, γ=0.6).

workflow (919,052), *namemaker* (106,839), *delete* (2,767), *deleteEnt* (2,354), *prpDelete* (471), *prpSmartDelete* (53), *prpModify* (34) and *cast* (1,430). Actors are classified as *user* (103), *viewer* (3) or *administrator* (2). In this Section, we illustrate the results obtained with a random sample of THINK3 data including 3580 executions. Process instances play the role of reference objects, while activities and actors play the role of task-relevant objects.

The goal of the experiments is to compare the conjunctive patterns discovered by SPADA with those disjunctive discovered by jSPADA in terms of the cardinality of the extracted patterns set and in terms of the learning time by varying threshold values. In the background knowledge D_I, a constant weight is assigned to each pair of activities. This way, the generation of a disjunctive form may equally involve each one of the activities. The weight is computed as the ratio of 1 (maximum dissimilarity value) to the number of distinct activities. In In THINK3, the weight is set 0.16 for each one of the six activities, while it is set to 0.33 for each one of the three actors.

Experiments are performed[2] by tuning the thresholds $minSup$, $nSup$ and γ (see Figure 2.a). As we expected, the number of final patterns (the summation of Conjunctive relational frequent patterns and Disjunctive relational frequent patterns) decreases as $minSup$ increases. Indeed, by enlarging the range $[nSup; minSup)$, the number of infrequent conjunctive patterns and the number of potential disjunctive patterns grow up, while the increase of $minSup$ leads to discover only those really frequent. Differently, narrowing the range $[nSup; minSup)$ leads to a larger set of Conjunctive frequent patterns but also to a smaller set of Conjunctive infrequent patterns which will be used in the pair-wise joining operation, and finally to a reduced set of disjunctive frequent patterns, as in the Figure 2.a when $minSup$=5%.

An interesting consideration is that the order of magnitude of the number of discovered disjunctive patterns is reasonably small (lower than 30). This permits to reach the objective of discovering process models with variants which are not difficult to interpret for the end-user. Another consideration can be found by the analysis of learning times (Figure 2.b) where it is possible to see that SPADA and jSPADA show comparable learning times. A deeper analysis reveals that by increasing $minSup$, the computational cost spent for the only generation of disjunctive patterns is of at least two orders of magnitude smaller than that of SPADA. Indeed, the increase of $minSup$ leads to enlarge the range $[nSup; minSup)$ and this would require longer learning time to process a greater set I_R. Actually, the reason of these time performances is twofold: first, a larger set I_R does not necessarily imply a larger set of disjunctive patterns given that, if the atoms of two patterns are different, no disjunction can be created; second, when $minSup$ has low values (e.g., 10%) the number of iterations in Algorithm 1 is smaller since disjunctive patterns support easily exceeds $minSup$.

A peculiarity of the approach is that it enriches relational patterns discovered by SPADA with additional atoms. For instance, the following pattern is discovered by jSPADA at $minSup$=20% ($nSup$=0.001,γ=0.6 in THINK3):

$$P_1 : \ process(A), \langle namemaker(A,B) \vee workflow(A,B) \rangle, user(B,C), \\ loggroup(C, ekm\_j) \qquad [support = 21.9\%]$$

P_1 states that activities $namemaker$ and $workflow$ occur one in alternative to the other. This happens in 785 executions out of 3850 executions where the actor is of kind $user$ and the login mode is ekm_j. P_1 joins the following two conjunctive patterns discovered by SPADA at $minSup$=5% that could represent two variants of the same model:

$$P_2 : process(A), namemaker(A,B), user(B,C), loggroup(C, ekm\_j) \quad [support = 8.1\%]$$

$$P_3 : process(A), workflow(A,B), user(B,C), loggroup(C, ekm\_j) \quad [support = 19.8\%]$$

Therefore, the proposed approach permits to unearth information extracted from SPADA at lower computational cost since the parameter $minSup$ strongly affects the learning times of both systems.

[2] Additional results are accessible at http://www.di.uniba.it/∼loglisci/jSPADA/.

5 Conclusions

In this paper, we present an approach to discover process models in the form of frequent relational patterns with disjunctions. Patterns describe activities and actors involved in the processes. Disjunctions permit to express $OR-split/OR-join$ constructs such that variants of process models can be identified. Experiments on real event logs empirically prove the effectiveness of the proposed approach. Discovered patterns permit to express variants and can be easily interpreted by end-users. As future work, we plan to investigate the applicability of relational frequent pattern mining in order to also mine loops.

Acknowledgment

This work is in partial fulfillment of the research objectives of the project ATENEO-2010: "Modelli e Metodi Computazionali per la Scoperta di Conoscenza in Dati Spazio-Temporali".

References

1. A. Appice, M. Ceci, A. Turi, and D. Malerba. A parallel, distributed algorithm for relational frequent pattern discovery from very large data sets. *Intelligent Data Analysis*, 15(1):69–88, 2011.
2. J. E. Cook and A. L. Wolf. Software process validation: Quantitatively measuring the correspondence of a process to a model. *ACM Trans. Softw. Eng. Methodol.*, 8(2):147–176, 1999.
3. S. Goedertier, D. Martens, B. Baesens, R. Haesen, and J. Vanthienen. A new approach for discovering business process models from event logs. In *Technical Report*. KBI 0716, 2007.
4. G. Greco, A. Guzzo, L. Pontieri, and D. Sacca'. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18:1010–1027, 2006.
5. S.-Y. Hwang, C.-P. Wei, and W.-S. Yang. Discovery of temporal patterns from process instances. *Comput. Ind.*, 53(3):345–364, 2004.
6. E. Lamma, P. Mello, F. Riguzzi, and S. Storari. Applying inductive logic programming to process mining. In *ILP 2007*, pages 132–146, 2007.
7. C. Li, M. Reichert, and A. Wombacher. Discovering reference models by mining process variants using a heuristic approach. In *BPM*, pages 344–362, 2009.
8. J. W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.
9. J. F. Roddick and P. Fule. Semgram - integrating semantic graphs into association rule mining. In *AusDM*, pages 129–137, 2007.
10. W. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, and C. Gnther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 9:87–111, 2010.
11. W. M. P. van der Aalst and A. J. M. M. Weijters. Process mining: a research agenda. *Comput. Ind.*, 53(3):231–244, 2004.
12. W. M. P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE TKDE*, 16(9):1128–1142, 2004.