Mining Expressive Process Models by Clustering Workflow Traces

Gianluigi Greco[1], Antonella Guzzo[1], Luigi Pontieri[2], and Domenico Saccà[1,2]

[1] DEIS, University of Calabria, Via Pietro Bucci 41C, 87036 Rende, Italy
[2] ICAR, CNR, Via Pietro Bucci 41C, 87036 Rende, Italy
{ggreco,guzzo}@si.deis.unical.it, {pontieri,sacca}@icar.cnr.it

Abstract. We propose a general framework for the process mining problem which encompasses the assumption of workflow schema with local constraints only, for it being applicable to more expressive specification languages, independently of the particular syntax adopted. In fact, we provide an effective technique for process mining based on the rather unexplored concept of clustering workflow executions, in which clusters of executions sharing the same structure and the same unexpected behavior (w.r.t. the local properties) are seen as a witness of the existence of global constraints.

An interesting framework for assessing the similarity between the original model and the discovered one is proposed, as well as some experimental results evidencing the validity of our approach.

1 Introduction

Even though workflow management systems (WfMS) are more and more utilized in enterprises, their actual impact in automatizing complex process is still limited by the difficulties encountered in the designing phase. In fact, processes have complex and often unexpected dynamics, whose modelling requires expensive and long analysis which may eventually result unviable under an economic viewpoint.

Recent research faced this problem, by exploiting some strategies, called *process mining* techniques, for using the information collected during the enactment of a process not yet supported by a WfMS, such as the transaction logs of ERP systems like SAP, in order to derive a model explaining the events recorded. Then, the output of these techniques, i.e., the "mined" synthetic model, can be profitably used to (re)design a detailed workflow schema, capable of supporting automatic enactments of the process.

Several approaches for process mining have been proposed in the literature (see, e.g., [1,16,4,12]), that aim at reconstructing the structure of the process, by exploiting graphical models based on the notion of *control flow graph*. This is an intuitive way of specifying a process through a directed graph, where nodes correspond to the activities in the process and edges represent the potential flow of work, i.e., the relationships of precedence among the activities.

However, despite its intuitiveness, the control flow completely lacks in the ability of formalizing complex *global constraints* on the executions, which often occurs while modelling real scenarios, for it being able to prescribe only *local constraints* in terms of relationships of precedence.

In this paper, we extend previous approaches to process mining, by proposing an algorithm which is able to discover not only the control flow of a given process, but also some interesting global constraints, in order provide the designer with a refined view of the process. The main contribution are as follows.

In Section 2, we formalize the process model discovery problem, in a context in which the target workflow schema may be enriched with some global constraints, denoted by $\mathcal{C}_G$. In order to decouple the approach from the particular syntax adopted for expressing $\mathcal{C}_G$, we exploit the observation that each global constraint leads to instances with a specific structure (short. pattern); then, a workflow schema $\mathcal{WS}^{\vee}$, accounting for global constraints, is the union of several schemas $\mathcal{WS}^1$, ..., $\mathcal{WS}^k$ (without global constraints), each one supporting the execution of one pattern, only.

Different patterns of executions (and, hence, $\mathcal{WS}^{\vee}$) are identified by means of an algorithm for clustering workflow traces, presented in Section 3, which is based on the projection of the traces on a suitable set of properly defined *features*. The approach is similar in the spirit to the proposals of clustering sequences using frequent itemsets, but technically more complex, for it deriving a hierarchical clustering. The theoretical properties of the algorithm are investigated as well.

In Section 3.1, we propose a level-wise algorithm for the identification of the set of features $\mathcal{F}$ for the clustering, and we study the problem of selecting the most 'representative' subset of $\mathcal{F}$, by showing its intrinsic difficulty. Therefore, we propose a greedy heuristic for quickly computing a set of features approximating the optimal solution.

Finally, we experiment an implementation of the proposed technique, by showing its scalability. An interesting framework for assessing the similarity between the original model and the discovered one is proposed in Section 4, thus, providing a quantitative way for testing the validity of the approach.

2 Formal Framework

In this section we formalize the mining problem addressed in the paper, which can be roughly described as the problem of (re)constructing a workflow model of an unknown process P, on the basis of log data related to some executions of the process.

The *control flow graph* of a process P is a tuple $\mathcal{CF}(P) = \langle A, E, a_0, F \rangle$, where A is a finite set of *activities*, $E \subseteq (A - F) \times (A - \{a_0\})$ is a relation of precedences among activities, $a_0 \in A$ is the starting activity, $F \subseteq A$ is the set of final activities.

Any connected subgraph $I = \langle A_I, E_I \rangle$ of the control flow graph, such that $a_0 \in A_I$ and $A_I \cap F \neq \emptyset$ is a *potential instance* of P. In order to model restrictions on the possible instances, the description of the process is often enriched with some additional *local* or *global* constraints, requiring, e.g., that an activity must (or may not) directly (or indirectly) follow the execution of a number of other activities.

For instance, local constraints are that an *and-join* activity can be executed only after all its predecessors are completed, and that an *or-join* activity can be executed as soon as one of its predecessors is completed. Other examples are than an *and-split* activity activates all of its successor activities, while a *xor-split* activates exactly one of its outgoing arcs.

Global constraints are, instead, richer in nature and their representation strongly depends on the particular application domain of the modelled process. Thus, they are often expressed using other complex formalisms, mainly based on a suitable logic with an associated clear semantics.

Let P be a process. A *workflow schema* for P, denoted by $\mathcal{WS}(P)$, is a tuple $\langle \mathcal{CF}(P), \mathcal{C}_L(P), \mathcal{C}_G(P)\rangle$, where $\mathcal{CF}(P)$ is the control flow graph of P, and $\mathcal{C}_L(P)$ and $\mathcal{C}_G(P)$ are sets of local and global constraints, respectively. Given a subgraph I of $\mathcal{CF}(P)$ and a constraint c in $\mathcal{C}_L(P) \cup \mathcal{C}_G(P)$, we write $I \models c$ whenever I satisfies c in the associated semantics. Moreover, if $I \models c$ for all c in $\mathcal{C}_L(P) \cup \mathcal{C}_G(P)$, I is called an *instance* of $\mathcal{WS}(P)$, denoted by $I \models \mathcal{WS}(P)$. When the process P is clear from the context, a workflow schema will be simply denoted by $\mathcal{WS} = \langle \mathcal{CF}, \mathcal{C}_L, \mathcal{C}_G\rangle$.

2.1 The Process Model Discovery Problem

Let A_P be the set of task identifiers for the process P. We assume the actual workflow schema $\mathcal{WS}(P)$ for P to be unknown, and we consider the problem of properly identifying it, in the set of all the possible workflow schemas having A_P as set of nodes. In order to formalize this problem we need some preliminarily definitions and notations.

A *workflow trace* s *over* A_P is a string in A_P^*, representing a task sequence. Given a trace s, we denote by $s[i]$ the $i\text{-}th$ task in the corresponding sequence, and by $lenght(s)$ the length of s. The set of all the tasks in s is denoted by $tasks(s) = \bigcup_{1 \leq i \leq lenght(s)} s[i]$. Finally, a *workflow log for* P, denoted by $\mathcal{L}_P$, is a bag of traces over Σ_P: $\mathcal{L}_P = [\ s \mid s \in A_P^*\ ]$ and is the only input from which inferring the schema $\mathcal{WS}(P)$.

In order to substantiate the problem of mining $\mathcal{WS}(P)$, one must specify which language is to be adopted for expressing the global constraints in $\mathcal{C}_G$. In order to devise a general approach, it is convenient to find an alternative (syntax-independent) way for evidencing global constraints. The solution adopted in this paper is to replace a unique target schema $\mathcal{WS}(P)$ with a variety of alternative schemata having no global constraints but directly modelling the various execution patterns prescribed by global constraints. The basic idea is to first derive from the trace logs an initial workflow schema whose global constraints are left unexpressed and, then, to stepwise refine it into a number of specific schemas, each one modelling a class of traces having the same characteristics w.r.t. global constraints.

Definition 1. Let P be a process. A *disjunctive workflow schema* for P, denoted by $\mathcal{WS}^\vee(P)$, is a a set $\{\mathcal{WS}^1, ..., \mathcal{WS}^m\}$ of workflow schemata for P, with $\mathcal{WS}^j = \langle \mathcal{CF}^j, \mathcal{C}_L^j, \emptyset\rangle$, for $1 \leq j \leq m$. The *size* of $\mathcal{WS}^\vee(P)$, denoted by $|\mathcal{WS}^\vee(P)|$, is the number of workflow schemata it contains. An instance of any $\mathcal{WS}^j$ is also an instance of $\mathcal{WS}^\vee$, denoted by $I \models \mathcal{WS}^\vee$. □

Given $\mathcal{L}_P$, we aim at discovering a disjunctive schema $\mathcal{WS}^\vee$ as "close" as possible to the actual unknown schema $\mathcal{WS}(P)$ that had generated the logs. This intuition can be formalized by accounting for two criteria, namely *completeness* and *soundness*, constraining the discovered workflow to admit exactly the traces of the log. Obviously, we preliminary need some mechanisms for deciding whether a given trace in $\mathcal{L}_P$ can be actually derived from a real instantiation of a workflow $\mathcal{WS}^\vee$. Ideally, we might exploit the following definition.

Definition 2. Let s be a trace in $\mathcal{L}_P$, $\mathcal{WS}^\vee$ be a disjunctive workflow schema, and $I = \langle A_I, E_I \rangle$ be an instance of it. Then, s is *compliant with* $\mathcal{WS}^\vee$ *through* I, denoted by $s \models^I \mathcal{WS}^\vee$, if s is a topological sort of I, i.e., s is an ordering of the activities in A_I s.t. for each $(a,b) \in E_I$, $i < j$ where $s[i] = a$ and $s[j] = b$. Moreover, s is simply said to be compliant with $\mathcal{WS}^\vee$, denoted by $s \models \mathcal{WS}^\vee$, if there exists I with $s \models^I \mathcal{WS}^\vee$. □

We are now ready to introduce, for a disjunctive workflow schema and for a trace log, the notions of soundness (i.e., every instance must be witnessed by some trace in the log) and of completeness (all traces are compliant with some instance). As the schema is not given but discovered from the analysis of the trace log, the two notions are given with a certain amount of uncertainty.

Definition 3. Let $\mathcal{WS}^\vee$ be a disjunctive workflow model, and $\mathcal{L}_P$ be a log for process P. We define:

- $soundness(\mathcal{WS}^\vee, \mathcal{L}_P) = \frac{|\{I | I \models \mathcal{WS}^\vee \wedge \nexists s \in \mathcal{L}_P \text{ s.t. } s \models^I \mathcal{WS}^\vee\}|}{|\{I | I \models \mathcal{WS}^\vee\}|}$, i.e., the percentage of instances having no corresponding traces in the log;
- $completeness(\mathcal{WS}^\vee, \mathcal{L}_P) = \frac{|\{s | s \in \mathcal{L}_P \wedge s \models \mathcal{WS}^\vee\}|}{|\{s | s \in \mathcal{L}_P\}|}$, i.e., the percentage of traces that are compliant with some trace in the log.

Given two real numbers α and σ between 0 and 1 (typically α is small whereas σ is close to 1) we say that $\mathcal{WS}^\vee$ is

- *α-sound* w.r.t. $\mathcal{L}_P$, if $soundness(\mathcal{WS}^\vee, \mathcal{L}_P) \leq \alpha$, i.e. the smaller the sounder;
- *σ-complete* w.r.t. $\mathcal{L}_P$, if $completeness(\mathcal{WS}^\vee, \mathcal{L}_P) \geq \sigma$, i.e., the larger the more complete. □

We want to discover a disjunctive schema $\mathcal{WS}^\vee$ for a given process P which is *α-sound* and *σ-complete*, for some given α and σ. However, it is easy to see that a trivial schema satisfying the above conditions always exists, consisting in the union of exactly one workflow (without global constraints) modelling each of the instances in $\mathcal{L}_P$. However, such model would be not a syntectic view of the process P, for its size being $|\mathcal{WS}^\vee| = |\mathcal{L}_P|$, where $|\mathcal{L}_P| = |\{s \mid s \in \mathcal{L}\}|$. We therefore introduce a bound on the size of $\mathcal{WS}^\vee$.

Definition 4. (Minimal Process Discovery) Let $\mathcal{L}_P$ be a workflow log for the process P. Given a real number σ and a natural number m, the *Minimal Process Discovery problem*, denoted by MPD(P,σ, m), consists in finding a σ-complete disjunctive workflow schema $\mathcal{WS}^\vee$, such that $|\mathcal{WS}^\vee| \leq m$ and $soundness(\mathcal{WS}^\vee, \mathcal{L}_P)$ is minimal. □

The problem is obviously solvable as one may sacrifice enough portions of soundness to get a result. But, as it is shown next, the problem is untractable. W.l.o.g., let us assume that the values representing soundness are suitably discretized as positive integers so that we can represent MPD as an NP optimization problem.

Theorem 1. MPD*(P,σ,m) is an* NP*-complete optimization problem whose set of feasible solution is not empty.*

Armed with the above result, we turn to the problem PD(P,σ,m) of greedily finding a suitable approximation, that is a σ-complete workflow schema $\mathcal{WS}^{\vee}$, with $|\mathcal{WS}^{\vee}| \leq m$, which is as sound as possible. In the rest, we shall propose an efficient technique for solving this problem.

3 Clustering Workflow Traces

In order to mine the underlying workflow schema of the process P (problem PD(P,σ,m)) we exploit the idea of iteratively and incrementally refining a schema, by mining some *global constraints* which are then used for discriminating the possible executions, starting with a preliminary disjunctive model $\mathcal{WS}^{\vee}$, which only accounts for the dependencies among the activities in P.

The algorithm `ProcessDiscover`, shown in Figure 1, which computes $\mathcal{WS}^{\vee}$ through a hierarchical clustering, first mines a control flow $\mathcal{CF}_{\sigma}$, according to the threshold σ[1], through the procedure *minePrecedences*, which mainly exploits techniques already presented in the literature (see, e.g., [1,18], and, therefore, it is not illustrated in more details. Each workflow schema $\mathcal{WS}_i^j$, eventually inserted in $\mathcal{WS}^{\vee}$, is identified by the number i of refinements needed, and an index j for distinguishing the schemas at the same refinement level. Moreover, we denote by $\mathcal{L}(\mathcal{WS}_i^j)$ the set of traces in the cluster defined by $\mathcal{WS}_i^j$. Notice that preliminarily $\mathcal{WS}_0^1$, containing all the logs in $\mathcal{L}_P$, is inserted in $\mathcal{WS}^{\vee}$, and in Step 3 we refine the model by mining some local constraints, too.

The algorithm is also guided by a greedy heuristic that at each step selects a schema $\mathcal{WS}_i^j \in \mathcal{WS}^{\vee}$, for being refined with the function *refineWorkflow*, by preferring the schema which can be most profitably refined. In practice, we refine the the least sound schema among the ones already discovered; however, some experiments have been also conduced refining the schema $\mathcal{WS}_i^j$ with the maximum value of $|\mathcal{L}(\mathcal{WS}_i^j)|$.

In order to reuse well know clustering methods, and specifically in our implementation the k-$means$ algorithm, the procedure *refineWorkflow* translates the logs $\mathcal{L}(\mathcal{WS}_i^j)$ to relational data with the procedures ***identifyRelevantFeatures*** and ***project***, which will be discussed in the next section. Then, if more than one feature is identified, it computes the clusters $\mathcal{WS}_{i+1}^{j+1}$, ..., $\mathcal{WS}_{i+1}^{j+k}$, where j is the maximum index of the schemas already inserted in $\mathcal{WS}^{\vee}$ at the level $i+1$, by applying the k-$means$ algorithm on the traces in $\mathcal{L}(\mathcal{WS}_i^j)$, and put inserts them into the disjunctive schema $\mathcal{WS}^{\vee}$. Finally, for each schema inserted $\mathcal{WS}^{\vee}$ the procedure *mineLocalConstraint* is applied, in order to identify local constraints as well.

The algorithm `ProcessDiscover` converges in at most m steps (see Step 4), and exploits the following interesting property of the procedure *refineWorkflow*. We observe that at each step of workflow refinement the value of soundness decreases, thus the algorithm gets closer to the optimal solution.

[1] Roughly, the edges in $\mathcal{CF}_{\sigma}$ represent a minimal set of precedences with at least a given support

```
Input: Problem PD($P$,$\sigma$,$m$), natural number $maxFeatures$.
Output: A process model.
Method: Perform the following steps:
 1 $\mathcal{CF}_\sigma(\mathcal{WS}_0^1)$ :=minePrecedences($\mathcal{L}_p$);        //See Section 3.1
 2 let $\mathcal{WS}_0^1$ be a schema, with $\mathcal{L}(\mathcal{WS}_0^1) = \mathcal{L}_P$;
 3 mineLocalConstraints($\mathcal{WS}_0^1$);        //See Section 3.1
 3 $\mathcal{WS}^\vee := \mathcal{WS}_0^1$;        //Start clustering with the dependency graph only
 4 while $|\mathcal{WS}^\vee| < m$ do
 5   $\mathcal{WS}_i^j$ :=leastSound($\mathcal{WS}^\vee$);
 6   $\mathcal{WS}^\vee := \mathcal{WS}^\vee - \{\mathcal{WS}_i^j\}$;
 7   refineWorkflow($i$,$j$);
 8 end while
 9 return $\mathcal{WS}^\vee$;
Procedure refineWorkflow($i$: step, $j$: schema);
 1 $\mathcal{F}$ :=identifyRelevantFeatures($\mathcal{L}(\mathcal{WS}_i^j), \sigma, maxFeatures, \mathcal{CF}_\sigma$);        //See Section 4.1
 2 $\mathcal{R}(\mathcal{WS}_i^j)$ :=project($\mathcal{L}(\mathcal{WS}_i^j), \mathcal{F}$);        //See Section 4.2
 3 $k := |\mathcal{F}|$;
 4 if $k > 1$ then
 5   $j := \max\{j \mid \mathcal{WS}_{i+1}^j \in \mathcal{WS}^\vee\}$;
 6   $\langle \mathcal{WS}_{i+1}^{j+1}, ..., \mathcal{WS}_{i+1}^{j+k} \rangle := k\text{-}means(\mathcal{R}(\mathcal{WS}_i^j))$;
 7   for each $\mathcal{WS}_{i+1}^h$ do
 8     $\mathcal{WS}^\vee = \mathcal{WS}^\vee \cup \{\mathcal{WS}_{i+1}^h\}$;
 9     $\mathcal{CF}_\sigma(\mathcal{WS}_{i+1}^h)$ :=minePrecedences($\mathcal{L}(\mathcal{WS}_{i+1}^h)$);
10     mineLocalConstraints($\mathcal{WS}_{i+1}^h$);
11   end for
12 else        //Leave of the tree
13   $\mathcal{WS}^\vee = \mathcal{WS}^\vee \cup \{\mathcal{WS}_i^j\}$;        //See Theorem 2.2
14 end if;
```

Fig. 1. **Algorithm** `ProcessDiscover`

Theorem 2. *Given a disjunctive schema $\mathcal{WS}^\vee$, with $\mathcal{WS}_i^j \in \mathcal{WS}^\vee$, the disjunctive workflow schema $\mathcal{WS}_+^\vee$, obtained by refining $\mathcal{WS}^\vee - \{\mathcal{WS}_i^j\}$ with the procedure refineWorkflow(i,j), is such that $soundness(\mathcal{WS}_+^\vee) \leq soundness(\mathcal{WS}^\vee)$.*

A main point of the algorithm is fixing the number k of new schemata to be added at each refinement step. The range of k goes from a minimum of 2, which will require several steps for the computation, to an unbounded value, which will return the result in only one step. One could then expect that the latter case is most efficient. This is not necessarily true: the clustering algorithm could run slower with a larger number of classes thus loosing the advantage of a smaller number of iterations. In contrast, there is an important point in favor of a small value for k: the representation of the various schemata can be optimized by preserving the tree structure and storing for each node only the differences w.r.t. the schema of the father node. The tree representation is relevant not only because of the space reduction but also because it give more insights on the properties of the modelled workflow instances and provides an intuitive and expressive description of global constraints.

3.1 Dealing with Relevant Features

The crucial point of the algorithm for clustering workflow traces lies in the formalization of the procedures ***identifyRelevantFeatures*** and ***project***. Roughly, the former identifies a set $\mathcal{F}$ of relevant features [10,11,14], whereas the latter projects the traces into a vectorial space whose components are, in fact, these features.

Some works addressing the problem of clustering complex data considered the most frequent common structures (see e.g. [2,3,7]), also called frequent patterns, to be the relevant features for the clustering. Since we are interested in features that witness some kind of global constraints, we instead exploit the more involved notion of unexpected (w.r.t. the local properties) frequent rules.

Let $\mathcal{L}$ be a set of traces, $\mathcal{CF}_\sigma$ be a mined control flow, for threshold σ, and E_σ be the edge set of $\mathcal{CF}_\sigma$. Then a sequence $[a_1...a_h]$ of tasks is σ-frequent in $\mathcal{L}$ if $|\{s \in \mathcal{L} \mid a_1 = s[i_1], ..., a_h = s[i_h] \wedge i_1 < ... < i_h\}|/|\mathcal{L}| \geq \sigma$. We say that $[a_1...a_h]$ *σ-precedes* a in $\mathcal{L}$, denoted by $[a_1...a_h] \dashrightarrow_\sigma a$, if both $[a_1...a_h]$ and $[a_1...a_h a]$ are σ-frequent in $\mathcal{L}$.

Definition 5 (Discriminant Rules). A *discriminant rule* (*feature*) ϕ is an expression of the form $[a_1...a_h] \not\dashrightarrow_\sigma a$, s.t. (i) $[a_1...a_h]$ is σ-frequent in $\mathcal{L}$, (ii) $(a_h, a) \in E_\sigma$, and (iii) $[a_1...a_h] \dashrightarrow_\sigma a$ does not hold. Moreover, ϕ is *minimal* if (iv) there is no b, s.t. $[a_1...a_h] \not\dashrightarrow_\sigma b$ and $[b] \dashrightarrow_\sigma a$, and (v) there is no j, s.t. $j > 1$ and $[a_j...a_h] \not\dashrightarrow_\sigma a$. □

The identification of discriminant rules can be carried out by means of the level-wise algorithm shown in Figure 2. At each step k of the computation, we store in L_k all the σ-frequent sequences whose size is k. Specifically, in the Steps 5–9, the set of potential sequences M to be included in L_{k+1} are obtained by combining those in L_k with the relationships of precedences in L_2 — notice that Step 7 prevents the computation of not minimal unexpected rules. Then, only σ-frequent pattern in M are included in L_{k+1} (Step 11), while all the others will determine unexpected rules (Step 12). The process is repeated until no other frequent traces are found. The correctness of the algorithm can be easily proven.

Theorem 3. *In the algorithm of Figure 2, before its termination (Step 16):*

1. *the set R contains exactly all the σ-frequent sequences of tasks, and*
2. *the set $\mathcal{F}$ contains exactly all the minimal discriminant rules.*

Notice that the algorithm *IdentifyRelevantFeatures* does not directly output $\mathcal{F}$, but call the procedure *mostDiscriminantFeatures*, whose aim is to find a proper subset of $\mathcal{F}$ which better discriminates the traces in the log.

This intuition can be formalized as follows. Let ϕ be a discriminant rule of the form $[a_i, ..., a_j] \not\dashrightarrow_\sigma b$, then *the witness of ϕ in $\mathcal{L}$*, denoted by $w(\phi, \mathcal{L})$, is the set of logs in which the pattern $[a_i, ..., a_j]$ occurs.

Moreover, given a set of rules R, then the witness of R in $\mathcal{L}$ is $\bigcup_{\phi \in R} w(\phi, \mathcal{L})$. For a fixed k, R is the most discriminant k-set of features if $|R| = k$ and there exists no R' with $|w(R', \mathcal{L})| > |w(R, \mathcal{L})|$, and $|R'| = k$. Notice that the most discriminant k-set of features can be computed in polynomial time by considering all the possible combinations of features of R, with k element.

The minimum k, for which the most discriminant k-set of features, say S, covers all the logs, i.e., $w(S, \mathcal{L}) = \mathcal{L}$, is called *dimension* of $\mathcal{L}$, whereas S is the *most discriminant set of features*.

```
Input: A log $\mathcal{L}$, a threshold $\sigma$, the max nr. of features $maxFeatures$, the control flow graph $\mathcal{CF}_\sigma$, with edge set $E_\sigma$.
Output: A set of minimal discriminant rules.
Method: Perform the following steps:
$L_2 := \{[ab] \mid (a,b) \in E_\sigma\}$;
$k := 1, R := L_2, \mathcal{F} := \emptyset$;
repeat
  $M := \emptyset; k := k + 1$;
  forall $[a_i...a_j] \in L_k$ do
    forall $[a_j b] \in L_2$ do
      if $[a_{i+1}...a_j] \not\dashrightarrow_\sigma b$ is not in $\mathcal{F}$ then
        $M := M \cup [a_i...a_j b]$;
  end for
  forall $p \in M$ of the form $[a_i...a_j b]$ do
    if $p$ is $\sigma$-frequent in $\mathcal{L}$ then $L_{k+1} := \{p\}$;
    else $\mathcal{F} := \mathcal{F} \cup \{[a_i...a_j] \not\dashrightarrow_\sigma b\}$;        //See Theorem 3.2
  end for
  $R := R \cup L_{k+1}$;        //See Theorem 3.1
until $L_{k+1} = \emptyset$;
return mostDiscriminant($\mathcal{F}$);

Procedure mostDiscriminantFeatures($\mathcal{F}$: set of unexpected rules): set of unexpected rules;
$S' := \mathcal{L}; \mathcal{F}' := \emptyset$;
do
  let $\phi = \texttt{argmax}_{\phi' \in \mathcal{F}} |w(\phi', S')|$;
  $\mathcal{F}' := \mathcal{F}' \cup \{\phi\}$;
  $S' := S' - w(\phi, S')$;
while $(|S'|/|\mathcal{L}_P| > \sigma)$ and $(\mathcal{F}' < maxFeatures)$;
return $\mathcal{F}'$;
```

Fig. 2. Algorithm *IdentifyRelevantFeatures*

Theorem 4. *Let $\mathcal{L}$ be a set of traces, n be the size of $\mathcal{L}$ (i.e., the sum of the lengths of all the traces in $\mathcal{L}$), and $\mathcal{F}$ be a set of features. Then, computing any most discriminant set of features is* NP *hard.*

Due to the intrinsic difficulty of the problem, we turn to the computation of a suitable approximation. In fact, the procedure *mostDiscriminantFeatures*, actually implemented in the algorithm for identifying relevant features, computes a set $\mathcal{F}'$ of discriminant rules, guided by the heuristics of greedily selecting a feature ϕ covering the maximum number of traces, among the ones (S') not covered by previous selections.

Finally, the set of relevant features $\mathcal{F}$, can be used for representing each trace s as a point in the vectorial space $\mathbb{R}^{|\mathcal{F}|}$, denoted by $\vec{s}$. Then, the procedure ***project*** maps traces in $\mathbb{R}^{|\mathcal{F}|}$, where k*-means* algorithm can operate. Due to its simplicity we do not report the code here.

4 Experiments

In this section we study the behavior of the `ProcessDiscover` algorithm for evaluating both its effectiveness and its scalability, with the help of a number of tests performed on synthetic data. The generation of such data can be tuned according to: (i) the size of $\mathcal{WS}$, (ii) the size of $\mathcal{L}_P$, (iii) the number of global constraints in $\mathcal{C}_G$, and (iv) the probability p of choosing any successor edge, in the case of *nondeterministic fork* activities. The ideas adopted in generating synthetic data are essentially inspired by [3], and the generator we exploited is an extension of the one described in [6].

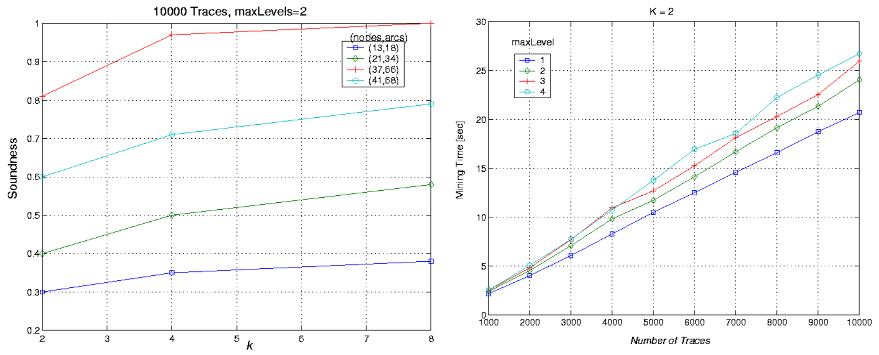

Fig. 3. Fixed Schema. Left: Soundness w.r.t. levels. Right: Scaling w.r.t. number of traces.

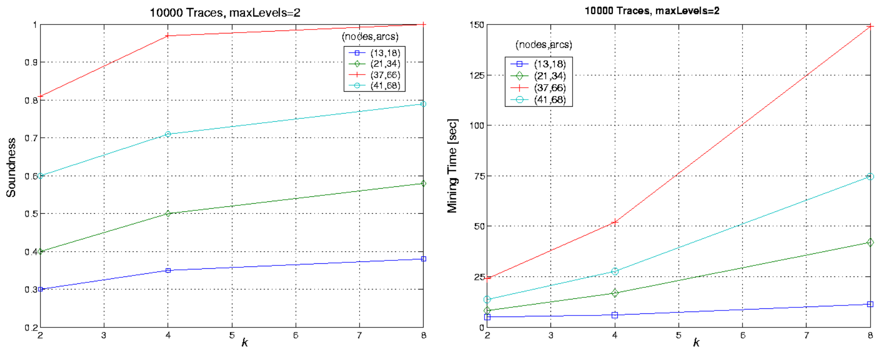

Fig. 4. Variable Workflow Schema. Left: Soundness w.r.t. k. Right: Scalability w.r.t. k.

Test Procedure. In order to asses the effectiveness of the technique, we adopted the following test procedure. Let $\mathcal{WS}(I)$ be a workflow schema for the input process I, and $\mathcal{L}_I$ a log produced with the generator. The quality of any workflow $\mathcal{WS}^\vee(O)$, extracted by providing the mining algorithm with $\mathcal{L}_I$, is evaluated, w.r.t. the original one $\mathcal{WS}(I)$, essentially by comparing two random samples of the traces they respectively admit. This allow us to compute an estimate of the actual soundness and completeness. Moreover, in order to avoid statistical fluctuations in our results, we generate a number of different training logs, and hence, whenever relevant, we report for each measure its mean value together with the associated standard deviation. In the test described here, we focus on the influence of two major parameters of the method: (i) the branching factor k and (ii) the maximum number ($maxLevels$) of levels in the resulting disjunctive scheme. Notice that the case $k = 1$ coincides with traditional algorithms which do not account for global constraints. All the tests have been conduced on a 1600MHz/256MB Pentium IV machine running *Windows XP Professional*.

Results. In a first set of experiments we considered a fixed workflow schema and some randomly generated instances. Figure 3 (on the left) reports the mean value and the standard deviation of the soundness of the mined model, for increasing values of $|\mathcal{L}_I|$

by varying the factor k. Notice that for $k = 1$, the algorithm degenerates in computing a unique schema, and in fact, the soundness is not affected by the parameter $maxLevel$ — this is the case of any algorithm accounting of local constraints only. Instead, for $k > 1$, we can even rediscover exactly the underlying schema, after a number of iterations. These experiments have been conduced on an input log of 1000 instances. Then, on the right, we report the scaling of the approach at the varying of the number of logs in $\mathcal{L}_I$.

In a second set of experiments we also consider variable schemas. In Figure 4 we report the results for four different workflow schemas. Observe (on the left) that for a fixed value of k, the soundness of the mined schema tends to be low at the increasing of the complexity of the schemas, consisting of many nodes and possibly many constraints. This witness the fact that on real processes, traditional approaches (with $k = 1$) performs poorly, and that for having an effective reconstruction of the process it is necessary not only to fix $k > 1$, but also to deal with several levels of refinements. Obviously, for complex schemas, the algorithm takes more time, as shown in the same figure on the right.

5 Conclusions

In this paper, we have continued on the way of the investigation of data mining techniques for process mining, by providing a method for discovering global constraints, in terms of the patterns of executions they impose. This is achieved through a hierarchical clustering of the logs, in which each trace is seen as a point of a properly identified space of features. The precise complexity of the task of constructing this space is provided, as well as a practical efficient algorithm for its solution.

References

1. R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proc. 6th Int. Conf. on Extending Database Technology (EDBT'98)*, pages 469–483, 1998.
2. R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of SIGMOD'93*, pages 207–216, 1993.
3. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proc. of the 20th Int'l Conference on Very Large Databases*, pages 487–499, 1994.
4. J.E.Cook and A.L. Wolf. Automating process discovery through event-data analysis. In *Proc. 17th Int. Conf. on Software Engineering (ICSE'95)*, pages 73–82, 1995.
5. H. Davulcu, M. Kifer, C.R. Ramakrishnan, and I.V. Ramakrishnan. Logic based modeling and analysis of workflows. In *Proc. of PODS'98*, pages 25–33, 1998.
6. G.Greco, A.Guzzo, G.Manco, and D. Saccà. Mining Frequent Instances on Workflows. In *Proc. of PAKDD'03*, pages 209–221, 2003.
7. J. Han, J. Pei, and Y. Yi. Mining frequent patterns without candidate generation. In *Proc. Int. ACM Conf. on Management of Data (SIGMOD'00)*, pages 1–12, 2000.
8. Y. Kim, W. Nick Street, and F. Menczer. Feature selection in unsupervised learning via evolutionary search. In *Proceedings of the KDD'00*, pages 365–369, 2000.
9. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR 163(4)* page 845-848, 1965.

10. N.Lesh, M.J. Zaki, and M.Ogihara. Mining features for sequence classification. In *Proc. of KDD'99*, pages 342–346, 1999.
11. H.Motoda and H.Liu. Data reduction: feature selection. pages 208–213, 2002.
12. P. Muth, J. Weifenfels, M.Gillmann, and G. Weikum. Integrating light-weight workflow management systems within existing business environments. In *Proc. 15th IEEE Int. Conf. on Data Engineering (ICDE'99)*, pages 286–293, 1999.
13. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proc. of ICDE'2001*, pages 215–224, 2001.
14. B.Padmanabhan and A.Tuzhilin. Small is beautiful: discovering the minimal set of unexpected patterns. In *Proc. of the 6th ACM SIGKDD*, pages 54–63, 2000.
15. W.M.P. van der Aalst. The application of petri nets to worflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
16. W.M.P. van der Aalst, A. Hirnschall, and H.M.W. Verbeek. An alternative way to analyze workflow graphs. In *Proc. 14th Int. Conf. on Advanced Information Systems Engineering*, pages 534–552, 2002.
17. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.
18. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G.Schimm, and A.J.M.M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.