Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

On the Mining of the Complex Workflow Schemas

Greco G., Guzzo A., Pontieri L., Saccà D.

RT-ICAR-CS-04-13 **Settembre2004**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

On the Mining of the Complex Workflow Schemas

G. Greco[1], A. Guzzo[1], L. Pontieri[2], D. Saccà[1,2]

Rapporto Tecnico N.:
RT-ICAR-CS-04-13

Data:
Settembre 2004

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Cosenza, Via P. Bucci 41C, 87036 Rende(CS)
[2] Università degli Studi della Calabria, DEIS, Via P. Bucci 41C, Rende (CS)

On the Mining of Complex Workflow Schemas

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

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

Abstract. Several recent works have addressed the problem of (re)discovering an unknown workflow model of a given process, by looking at the logs of a number of its executions. Most of these approaches assume a graphical representations of the model, namely the *control flow graph*, which provides an intuitive description of the precedence relationships between the underlying activities, often enriched with some kind of *local constraints*, such as synchronization or parallel executions. In this paper we extend such approaches by proposing 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 process mining technique 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 trace of the existence of global constraints.

1 Introduction

Process mining techniques, i.e., techniques that look at the information collected during the enactment of a process, not yet supported by a Workflow Management System, in order to derive a model explaining the event recorded, are increasingly exploited within enterprises. In fact, they can be fruitfully exploited to assist the administrator in the design of processes with complex and often unexpected dynamics, whose modelling with traditional approaches would require expensive and long analysis which may eventually result unviable under an economic viewpoint.

As for a typical applicative scenario, we shall consider throughout the paper the automatization of the (*OrderManagement*) process of handling customers' orders within a business company, consisting of the activities of (`a`) receiving an order, (`b`) authenticating the client, (`c`) checking in stock the availability of the required product, (`d`) verifying the availability of external supplies, (`f`) registering a client in the company database, (`i`) evaluating the trustworthiness of the client, (`g`) evaluating the plan of the production, (`h`) rejecting an order, (`l`) accepting an order, (`n`) preparing the bill, (`m`) applying discount for regular customers, and (`o`) contacting the mail department in order to speed up the shipment of the goods.

In this scenario, the workflow designer might only have a look at some execution traces, like the ones shown in Table 1, and then she can use some of the process mining approaches proposed in the literature (see, e.g., [1, 15, 4, 12]), aiming at reconstructing

s_1 : acdbfgih	s_5 : abicglmn	s_9 : abficgln	s_{13} : abcidglmn
s_2 : abficdgh	s_6 : acbiglon	s_{10} : acgbfilon	s_{14} : acdbiglmn
s_3 : acgbfih	s_7 : acbgilomn	s_{11} : abcfdigln	s_{15} : abcdgilmn
s_4 : abcgiln	s_8 : abcfgilon	s_{12} : acdbfigln	s_{16} : acbidgln

Table 1. Sample log traces from the process *OrderManagement*

the structure of the process. Most of these approaches exploit graphical models based on the notion of *control flow graph*, which describe the process through a directed graph, where nodes correspond to the activities and edges represent the potential flow of work, i.e., the precedence relationships between activities. However, despite its intuitiveness, the control flow completely lacks in the ability of formalizing complex *global constraints* on the executions, often occurring in real scenarios, since it only can prescribe *local constraints* in terms of precedence relationships.

In this paper, we extend previous approaches to process mining, by proposing an algorithm which, besides discovering the control flow of a process, can also reveal the presence of interesting global constraints, so providing the designer with a refined view of the process. In order to substantiate the problem, one should specify the language to be adopted for expressing global constraints — thus the problem is strongly dependent on syntactical issues. Therefore, in order to devise a general approach, in Section 3 we find an alternative (syntax-independent) way for evidencing global constraints, consisting in replacing a unique target schema $\mathcal{WS}$ with a set of alternative schemata having no global constraints but directly modelling the execution patterns those constraints prescribe.

Different patterns of executions are identified by means of an algorithm for *hierarchically* clustering workflow traces, presented in Section 4. In order to reuse well known clustering methods, the algorithm exploits a “flat”, relational representation of the traces, obtained by projecting the instances onto a set of suitably defined *features*. Specifically, each feature is a frequent structure of execution which is discovered by means of proper data mining techniques. Thus, the approach is similar in the spirit to the proposals (see, e.g., [10]) of clustering sequences using the frequent itemsets as relevant features, but technically more complex, for it deriving a hierarchical clustering for workflow schemata, whose structure is more complex than simple sequences.

We conclude by stressing that all the techniques presented here have been implemented and quantitatively tested by exploiting an interesting framework for assessing the similarity between the original model and the discovered one. The results of such an experimentation are discussed in Section 5.

2 Related Work

In this section, we next briefly review some recent advances in the application of data mining techniques within the context of Workflow Management Systems.

The first approach to process mining, in a Software Engineering setting, is introduced in [4], where three different methods are proposed to automatically derive a formal process model from execution’s log. All these methods use an event stream, i.e., a sentence in a three-token language, as input for inferring a Finite State Machine (FSM) model which reflects the behavior of the process. In such a representation language, the activities, corresponding to input tokens, are represented by edges and specify transitions between states of the underlying process.

The application of process mining to the workflow management context is firstly investigated in [1]. Differently from the previous approach, based on the finite state machine model, the authors adopt a *directed graph model.* The main restriction of such model is that it does not consider either edges labelled or AND/OR of joins and splits. Actually, in this model a dependency between two activities can be only represented as either a directed edge or a path from an activity to the other one. Conversely, our model is able to express richer control flow constructs, for specifying concurrency, synchronization and choices.

In [16] a special kind of Petri nets, named Workflow nets (WF-net), is used to model the control flow of a process. Tasks are modelled by transitions and causal dependencies are modelled by places and arcs. In particular, a place corresponds to a condition which can be used as pre- or post-condition for tasks. Such a model, which can be used to specify the execution of a workflow instance, allows for expressing complex control flow constraints. The mining task consists in deriving a WF-net from a workflow log (containing just sequences of task executions, as usual), under the assumption that the log is complete, in the sense that every causal relationship between tasks is witnessed by the consecutive appearance of those tasks in at least one trace of the log. The approach relies on the computation of some simple statistics on the consecutive appearance of tasks in the log, from which a preliminary (acyclic) causal graph is derived. Such a model is possibly enriched by introducing cycles (recursion and short-loop) and by determining the nature of split and join nodes, still on the basis of the previously computed statistics. The final result of the algorithm is a suitable WF-net which encodes all information on the behavior of the process extracted from the log.

Finally, a strictly related but quite different data mining problem in workflow management context has been introduced in [6]. Rather than using logs for mining a process model, the authors assume that a model is already know and consider the ability of predicting the “most probable” workflow execution. Indeed, in real world-cases, many choices can be done during a workflow execution: some of them can lead to a benefit, other ones should be avoided in the future. Data mining techniques can help in taking the appropriate decisions during in the execution of further coming instances, by extracting unexpected and useful knowledge about the process from previous instantiations of the workflow itself. The algorithm presented in [6] is able to discover the connected structure of the executions that have been scheduled more frequently by the workflow system, i.e., whose frequency of occurrence in the logs is above a given threshold σ. These structures are simply called *frequent connected patterns* (short: frequent $\mathcal{F}$-patterns) and they are the subprocesses that are frequently performed during the enactment of the main process.

3 Formal Framework

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

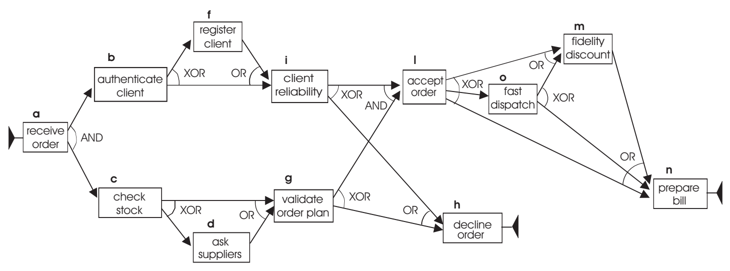

Fig. 1. Control flow graph for the sample *OrderManagement* process.

activities, $a_0 \in A$ is the starting activity, $F \subseteq A$ is the set of final activities. For instance, Figure 1 shows a possible control flow for the *OrderManagement* process presented in the Introduction.

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.

Most of the approaches proposed in the literature, even though with possibly different syntaxes, assume that the local constraints can be expressed in terms of three functions $\texttt{IN}, \texttt{OUT}_{min}$, and $\texttt{OUT}_{max}$ assigning to each node a natural number ($A \mapsto \mathbb{N}$) as follows:

– $\forall a \in A - \{a_0\}$, $0 < \texttt{IN}(a) \leq \mathit{InDegree}(a)$;
– $\forall a \in A - F$, $0 < \texttt{OUT}_{min}(a) \leq \texttt{OUT}_{max}(a) \leq \mathit{OutDegree}(a)$;
– $\texttt{IN}(a_0) = 0$, and $\forall a \in F$, $\texttt{OUT}_{min}(a) = \texttt{OUT}_{max}(a) = 0$.

where $\mathit{InDegree}(a) = |\{e = (b, a) \mid e \in E\}|$ and $\mathit{OutDegree}(a) = |\{e = (a, b) \mid e \in E\}|$.

As for the semantics, an activity a can start as soon as at least $\texttt{IN}(a)$ of its predecessor activities have been completed. Two typical cases are: (i) if $\texttt{IN}(a) = \mathit{InDegree}(a)$ then a is an *and-join* activity, for it can be executed only after all of its predecessors are completed, and (ii) if $\texttt{IN}(a) = 1$ then a is an *or-join* activity, for it can be executed as soon as one of its predecessors is completed.

Once finished, an activity a activates one non-empty subset of its outgoing arcs with cardinality between $\texttt{OUT}_{min}(a)$ and $\texttt{OUT}_{max}(a)$. If $\texttt{OUT}_{max}(a) = \mathit{OutDegree}(a)$ then a is a *full fork* and if also $\texttt{OUT}_{min}(a) = \texttt{OUT}_{max}(a)$ then a is a *deterministic fork* (also known as "and-split"), for it activates all of its successor activities. Finally, if $\texttt{OUT}_{max}(a) = 1$ then a is an *exclusive fork* (also called *xor-split* in the literature), for it 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 $\mathcal{WS} = \langle \mathcal{CF}, \mathcal{C}_L, \mathcal{C}_G \rangle$.

3.1 The Process Model Discovery Problem

Let A_P be the set of task identifiers for 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 sequence of activities. Given a trace s, we denote by $s[i]$ the *i-th* task in the corresponding sequence, and by *lenght*(s) the length of s. The set of all the activities 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 $\mathcal{WS}(P)$. Notice that any trace s is indeed a topological sort of some instance $I = \langle A_I, E_I \rangle$ of $W(P)$, 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 $s[j] = b$ – incidentally, we say that I is an instance corresponding to the trace s and that s is a trace corresponding to the instance I.

The basic idea for mining global constraints is, first, to 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 schemata, each one modelling a class of traces having the same characteristics w.r.t. global constraints. 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 generated the log traces. This intuition can be formalized by accounting for two criteria, namely *completeness* (i.e., all traces are compliant with some instance) and *soundness* (i.e., every instance must be witnessed by some trace in the log), constraining the discovered workflow to admit exactly the traces of the log.

Let *soundness*$(\mathcal{WS}^\vee, \mathcal{L}_P)$ denote the percentage of instances of $\mathcal{WS}^\vee$ having no corresponding traces in the log, and *completeness*$(\mathcal{WS}^\vee, \mathcal{L}_P)$ denote the percentage of traces in $\mathcal{L}_P$ for which there are corresponding instances of $\mathcal{WS}^\vee$. Then, 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. Moreover, $\mathcal{WS}^\vee$ is *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.

Our aim is We 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 of the union of exactly one workflow schema (without global constraints) for each of the instances in $\mathcal{L}_P$. Nonetheless, 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 $\mathcal{WS}^\vee$, and we propose the following problem.

Definition 1. (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. □

It can be shown that in the above problem cannot be efficiently solved (unless, $P =$ NP). Thus, in the paper, we shall propose an efficient technique for solving the strictly-related 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.

4 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 2, which computes $\mathcal{WS}^\vee$ through a hierarchical clustering, first mines a control flow $\mathcal{CF}_\sigma$,[1] through the procedure *minePrecedences* according to the threshold σ, which mainly exploits techniques reported in Section 4.1. 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 $\mathcal{WS}_i^j \in \mathcal{WS}^\vee$, to be refined through 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, where the schema $\mathcal{WS}_i^j$ with the maximum value of $|\mathcal{L}(\mathcal{WS}_i^j)|$ is chosen.

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 Sections 4.2 and 4.3, respectively. 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 them in the disjunctive schema $\mathcal{WS}^\vee$. Finally, for each schema inserted in $\mathcal{WS}^\vee$ the procedure *mineLocalConstraint* is applied, in order to identify local constraints as well. It can be shown 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  $\mathcal{F}$ :=$\boldsymbol{identifyRelevantFeatures}(\mathcal{L}(\mathcal{WS}_i^j), \sigma, maxFeatures, \mathcal{CF}_\sigma)$;          //See Section 4.1
 2  $\mathcal{R}(\mathcal{WS}_i^j)$ :=$\boldsymbol{project}(\mathcal{L}(\mathcal{WS}_i^j), \mathcal{F})$;          //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    $\langle \mathcal{WS}_{i+1}^{j+1}, ..., \mathcal{WS}_{i+1}^{j+k} \rangle$ := $k$-$means(\mathcal{R}(\mathcal{WS}_i^j))$;
 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. 2. Algorithm `ProcessDiscover`

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.

4.1 Dependencies and Local Constraints

In this section we overview and extend some ideas proposed in the literature, for mining both dependencies and local constraints. First, we need a notion for inferring precedences among set of activities w.r.t. their actual occurrences in the log.

Let $\mathcal{L}_P$ be a workflow log over Σ_P, $A \subseteq \Sigma_P$ be a set of activities, and s a trace in $\mathcal{L}_P$. The *beginning* (resp. *ending*) of A in s, denoted by $b(A, s)$ (resp. $e(A.s)$), is the index i, if exists, such that $a = s[i]$, and $\forall a' \in A - \{a\}$, $a' = s[j]$ with $j > i$ (resp. $j < i$). Given

$B \subseteq \Sigma_P$, and a threshold σ, we say that A *σ-precedes* B in $\mathcal{L}_P$, denoted by $A \rightarrow_\sigma B$, if $|\{s \in \mathcal{L}_P \mid e(A,s) < b(B,s)\}|/|\mathcal{L}_P| \geq \sigma$.

Exploiting this notions, we can characterize complex relationships among tasks. Given two activities a and b, and a threshold σ, we say that:

– a and b are *σ-parallel activities* in $\mathcal{L}_P$, denoted by $a \|_\sigma b$, if there are activities $a = a_1, ..., b = a_m$ with $m > 1$ such that $\{a_i\} \rightarrow_\sigma \{a_{i+1}\}$ for $1 \leq i < m$, and $\{a_m\} \rightarrow_\sigma \{a_1\}$.
– a *σ-strictly precedes* b in $\mathcal{L}_P$, denoted by $a \Rightarrow_\sigma b$, if a and b are not *σ-parallel activities*, and if there are traces $s_1, ..., s_k$ in $\mathcal{L}_P$, with $k = \sigma \times |\mathcal{L}_P|$, such that for each s_i, $b(\{a\}, s_i) < b(\{b\}, s_i)$, and $\forall j$ s.t. $b(\{a\}, s_i) < j < b(\{b\}, s_i)$, $s_i[j] \|_\sigma b$.

Parallel activities and strictly precedences are the basic blocks from which the control flow is inferred. Indeed, the *σ-control flow* of P is the graph $\mathcal{CF}_\sigma(P) = \langle \Sigma_P, E_\sigma \rangle$ containing an arc (a, b) in E_σ for each pair of nodes a and b, such that either (i) $a \Rightarrow_\sigma b$ or (ii) $\{a\} \rightarrow_\sigma \{b\}$ and does not exist a set of activities $\{h_1, ..h_m\}$ with $a \Rightarrow_\sigma h_1$, $h_i \Rightarrow_\sigma h_{i+1}$ for $1 \leq i < m$, and $h_m \Rightarrow_\sigma b$.

Finally, the set of *σ-local constraints*, denoted by $\mathcal{C}_{L\sigma}$, can be mined by exploiting the control flow:

$$\begin{aligned}
&\texttt{OUT}_{min}(a) = |succ(a)| - \max_{S \subseteq succ(a), \{a\} \not\rightarrow_\sigma S} |S| \\
&\texttt{OUT}_{max}(a) = |succ(a)| - \min_{S \subseteq succ(a), \{a\} \not\rightarrow_\sigma S} |S| \\
&\texttt{IN}(a) = \min_{S \subseteq prec(a), S \rightarrow_\sigma \{a\}} |S|
\end{aligned}$$

where $succ(a) = \{b \mid (a, b) \in E_\sigma\}$ and $prec(a) = \{b \mid (b, a) \in E_\sigma\}$, and $A \not\rightarrow_\sigma B$ simply denotes that relation $A \rightarrow_\sigma B$ does not hold.

4.2 Dealing with Relevant Features

A crucial point in the algorithm for clustering workflow traces is 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 onto 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, 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 $[a_1...a_h] \rightarrow_\sigma a$, if both $[a_1...a_h]$ and $[a_1...a_h a]$ are σ-frequent in $\mathcal{L}$.

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) $[a_1...a_h] \rightarrow_\sigma a$ does not hold. Moreover, ϕ is *minimal* if (iv) there is no b, s.t. $[a_1...a_h] \not\dashrightarrow_\sigma b$ and $[b] \rightarrow_\sigma a$, and (v) there is no j, s.t. $j > 1$ and $[a_j...a_h] \not\dashrightarrow_\sigma a$.

```
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:
 1    $L_2 := \{[ab] \mid (a,b) \in E_\sigma\}$;
 2    $k := 1$, $R := L_2$, $\mathcal{F} := \emptyset$;
 3    repeat
 4       $M := \emptyset$; $k := k+1$;
 5       forall $[a_i...a_j] \in L_k$ do
 6         forall $[a_j b] \in L_2$ do
 7            if $[a_{i+1}...a_j] \not\dashrightarrow_\sigma b$ is not in $\mathcal{F}$ then
 8               $M := M \cup [a_i...a_j b]$;
 9       end for
10       forall $p \in M$ of the form $[a_i...a_j b]$ do
11          if $p$ is $\sigma$-frequent in $\mathcal{L}$ then $L_{k+1} := \{p\}$;
12          else $\mathcal{F} := \mathcal{F} \cup \{[a_i...a_j] \not\dashrightarrow_\sigma b\}$;          //See Theorem 3.2
13       end for
14    $R := R \cup L_{k+1}$;          //See Theorem 3.1
15    until $L_{k+1} = \emptyset$;
16    return $mostDiscriminant(\mathcal{F})$;
```

```
Procedure mostDiscriminantFeatures($\mathcal{F}$: set of unexpected rules): set of unexpected rules;
 1 $S' := \mathcal{L}$; $\mathcal{F}' := \emptyset$;
 2 do
 3    let $\phi = \mathtt{argmax}_{\phi' \in \mathcal{F}} |w(\phi', S')|$;
 4    $\mathcal{F}' := \mathcal{F}' \cup \{\phi\}$;
 5    $S' := S' - w(\phi, S')$;
 6 while $(|S'|/|\mathcal{L}_P| > \sigma)$ and $(\mathcal{F}' < maxFeatures)$;
 7 return $\mathcal{F}'$;
```

Fig. 3. Algorithm *IdentifyRelevantFeatures*

For instance, in the *OrderManagament* process, $[fil] \not\dashrightarrow_{5/16} m$ is a minimal discriminant rule, witnessing the global constraint that fidelity discount is not applied for new clients. Notice that $[dgl] \not\dashrightarrow_{5/16} o$ is a minimal discriminant rule as well.

The identification of discriminant rules can be carried out by means of the level-wise algorithm shown in Figure 3. 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 1. *In the algorithm of Figure 3, 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 it calls instead the procedure *mostDiscriminantFeatures*, searching for a proper subset of $\mathcal{F}$ which discriminates the log traces at best.

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

Traces	s_1... s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}
s_{ϕ_1}	0 ... 0	0.8212	0.9018	0.8212	0.9018	0.9018	0	0	0	0
s_{ϕ_2}	0 ... 0	0	0	0	0.9018	0.9018	0.8212	0.8212	0.8212	0.9018

Table 2. Sample traces from *OrderManagement* in the feature space

$|w(R', \mathcal{L})| > |w(R, \mathcal{L})|$, 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.*

Theorem 2. *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.

4.3 Projecting Traces

The last aspect to overview for the algorithm `ProcessDiscover`, shown in Figure 2, concerns the way in which the selected features can be used for mapping traces in a proper vectorial space, where *k-means* algorithm can be applied. This is carried out within the procedure ***project***. Due to its simplicity we do not report the code here, and we give just a flavor of its behavior.

The main idea is that 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 $\overrightarrow{s}$.

Let $\phi : [a_1...a_h] \not\dashrightarrow a$ be a feature in $\mathcal{F}$, then the value assigned to the component of $\overrightarrow{s}$ associated with ϕ is 0 if $\{a_1, ..., a_h\} \not\subseteq tasks(s)$, and $(1 + \alpha(s, \phi) \times \beta(s, \phi)^2)^{-1}$ otherwise. The coefficient $\alpha(s, \phi)$ represents an empirical (inverse) estimation of the impact feature ϕ could have in pruning the space of the possible execution branches when producing trace s. Conversely, $\beta(s, \phi)$ represents a correction factor, which takes into account the number of other activities which occurred in s before or in between those of ϕ, and which could have influenced the sequel of the execution as well:

$$\alpha(s, \phi) = \frac{|\ F_G(a_h) \cap \{\ t \in tasks(s)\ |\ s[i]=t \wedge i \geq e(\{a_h\}, s)\ \}\ |}{|\ F_G(a_h)\ |} \quad ; \quad \beta(s, \phi) = \frac{b(\{a_h\}, s)\ -\ |\ tasks(body(\phi))\ |}{b(\{a_h\}, s)}$$

where $F_G(a)$ is the set of tasks which can be reached from a in G, whereas the functions e and b hold the same meaning as in the previous section.

Table 2 reports the results of projecting the sample traces in Table 1 over the features $\phi_1 = [bfil] \not\dashrightarrow m$ and $\phi_2 = [cdgl] \not\dashrightarrow o$.

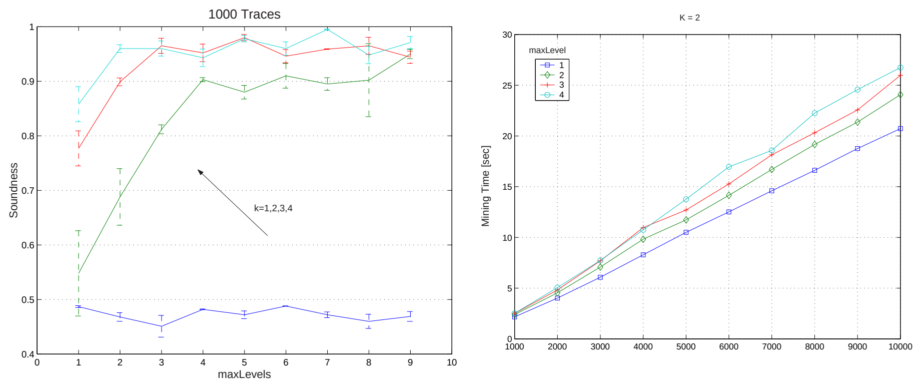

Fig. 4. Fixed Schema. Left: Soundness w.r.t. levels. Right: Scaling w.r.t. number of traces.

5 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].

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*.

We considered a fixed workflow schema and some randomly generated instances. Figure 4 (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. Finally, on the right, the graph shows that our approach scales (almost) linearly at the varying of the number of logs in $\mathcal{L}_I$.

6 Conclusions

In this paper, we have continued on the way of investigating data mining techniques for process analysis, 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, where each trace is seen as a point in a properly identified feature space. The complexity of the task of constructing this space is provided, as well as a viable algorithm for its solution. We conclude by mentioning that a problem that we did not address in this paper is how to handle the presence of noise on the logs, due to erroneous insertions or non-insertions of activities or bad reporting of order time sequence. A promising solution is to introduce a weak notion of trace based on the edit distance of two strings [9].

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, 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.
16. 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.