Discovering Expressive Process Models by Clustering Log Traces

Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Saccà, *Member*, *IEEE Computer Society*

Abstract—Process mining techniques have recently received notable attention in the literature for their ability to assist in the (re)design of complex processes by automatically discovering models that explain the events registered in some log traces provided as input. Following this line of research, the paper investigates an extension of such basic approaches, where the identification of different variants for the process is explicitly accounted for, based on the clustering of log traces. Indeed, modeling each group of similar executions with a different schema allows us to single out "conformant" models, which, specifically, minimize the number of modeled enactments that are extraneous to the process semantics. Therefore, a novel process mining framework is introduced and some relevant computational issues are deeply studied. As finding an exact solution to such an enhanced process mining problem is proven to require high computational costs, in most practical cases, a greedy approach is devised. This is founded on an iterative, hierarchical, refinement of the process model, where, at each step, traces sharing similar behavior patterns are clustered together and equipped with a specialized schema. The algorithm guarantees that each refinement leads to an increasingly sound model, thus attaining a monotonic search. Experimental results evidence the validity of the approach with respect to both effectiveness and scalability.

Index Terms—Process mining, data mining, workflow management, clustering, classification, association rules.

1 Introduction

Even though workflow management systems are ever more utilized in enterprises, their actual impact in managing complex processes is still limited by the difficulties encountered in the design phase. Indeed, processes often have complex dynamics whose modeling requires expensive and long analysis, which may be eventually unfeasible from an economic viewpoint. Therefore, several approaches have been recently proposed in the literature to accommodate the design of complex workflows by means of *process mining* techniques (see Section 7, for an overview of different proposals).

These approaches are devoted to automatically deriving a model that can explain all the episodes recorded in the event *log* (also known as transactional log or audit trail [1]), which is, in fact, collected and stored by most information systems while the activities of a given process are executed—this is the case of transactional information systems such as Workflow Management (WFM), Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Business to Business (B2B), and Supply Chain Management (SCM) systems. Eventually, the "mined" model is used to (re)design a detailed workflow schema, capable of supporting forthcoming enactments.

- G. Greco is with the Department of Mathematics, University of Calabria, Via Bucci 30B, I87036, Rende (CS), Italy. E-mail: ggreco@mat.unical.it.
- A. Guzzo and D. Saccà are with ICAR-CNR and DEIS, University of Calabria, Via Bucci 41C, I87036 Rende (CS), Italy. E-mail: guzzo@icar.cnr.it, sacca@unical.it.
- L. Pontieri is with the Institute of High Performance Computing and Networks (ICAR-CNR), Via Bucci 41C, I87036 Rende (CS), Italy. E-mail: pontieri@icar.cnr.it.

Manuscript received 24 June 2005; revised 24 Mar. 2006; accepted 11 Apr. 2006; published online 19 June 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0245-0605.

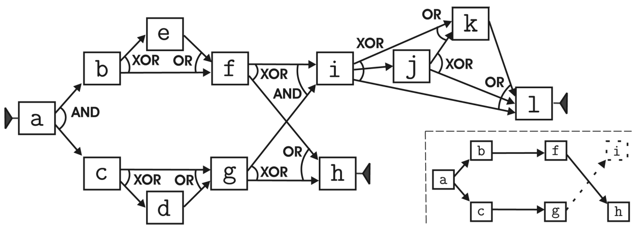
Example 1.1. As a sample applicative scenario, throughout the paper the process of handling customers' orders within a business company will be considered. The process (called *OrderManagement*) consists of the following activities:

a. receiving the order,
b. authenticating the client,
c. checking the availability of the required product in the stock,
d. verifying the availability of external supplies,
e. registering the client in the company database,
f. evaluating the trustworthiness of the client,
g. evaluating the production plan,
h. rejecting the order,
i. accepting the order,
j. contacting the mail department in order to speed up shipment of the goods,
k. applying some discount, and
l. preparing the bill.

In this scenario, some execution traces may be available for they are collected by the information systems of the business company (e.g., the log in Fig. 1a). And, in fact, the aim of process mining techniques is to look at these traces and to automatically identify a model for the *OrderManagement* process.

Actually, since several alternative models may be singled out for any fixed log, techniques have been proposed for identifying the most *conformant* one, i.e., the model which is the best "aligned" with the log. In this respect, the most widely considered measure for conformance is the *completeness* of the model (similar to *fitness* in [2]) which is, roughly speaking, the percentage of the *traces* stored in the log that

s_1 :	acbgfh	s_9 :	abefcgil
s_2 :	abfcgh	s_{10} :	acgbefijl
s_3 :	acgbfh	s_{11} :	abcedfgil
s_4 :	abcgfil	s_{12} :	acdbefgil
s_5 :	abfcgikl	s_{13} :	abcfdgikl
s_6 :	acbfgijl	s_{14} :	acdbfgikl
s_7 :	acbgfijkl	s_{15} :	abcdgfikl
s_8 :	abcegfijl	s_{16} :	acbfdgil



(a) (b) (c)

Fig. 1. An example process: (a) some traces, (b) workflow schema $\mathcal{WS}_0$, and (c) an example instance.

may be the result of some enactment supported by the mined model. Then, differences among various proposals for process mining often come in the language used for representing process models and in the specific algorithms used for mining the best-fitting one.

In this paper, progress toward enhancing the process mining framework is made, specifically by taking care of the following two key issues:

1. Good completeness of the model does not necessarily entail its conformance with the underlying process P. For instance, a fully complete model may support not only all the traces provided in input, but also an arbitrary number of execution patterns that are not registered in the log. In many cases, such a situation might strongly undermine the significance of the mined model with regards to the application needs. Therefore, models that should be focused on it are not too generic and they appropriately encode the behavior of the process. This can be accomplished by taking care of another parameter, called *soundness* (similar to *minimality* or *behavioral appropriateness* in [2]), which measures the percentage of enactments of the mined model that do find some correspondence in the log. Intuitively, the lower the factor, the more extraneous (with regard to the process semantics) executions the model supports.
2. The output of process mining techniques is usually too complex to be practically exploited by the designer, as emerged in several real applications domains (see, e.g., [3], [4]). This aspect is especially relevant in the case of processes consisting of a large number of activities, with several specific execution-scenarios and complex behavioral rules, for which the mined models may be made up of hundreds of tasks and constraints. Nonetheless, very little effort has been made to support some kinds of abstraction (e.g., techniques in [5], [6], [7] are able to produce models that focus on the main behavior as reported in the log by properly dealing with noise). In this respect, a viable solution for a better understanding of the process semantics is to take care of the existence of variants of the process, so that each variant may be independently mined and equipped with a specialized model. Clearly, only the most relevant variants should be identified in order to prevent the model from becoming too specific. Then, suitable strategies supporting some kinds of trade-off in the identification of the variants and, possibly, the ability of incrementally refining the model should be conceived.

Though some of the approaches in the literature already exploit some minimality conditions for increasing the soundness (e.g., [8], [9]), very little effort has been made to discover variants and usage scenarios (cf. [10], [11]). However, the isolation of the variants would make clear the "implicit" model (cf. [9]) underlying a process, i.e., the knowledge (organizational/behavioral rules) that actors have in mind during the enactments. Specifically, if variants for the process are not discovered and properly classified, the mined model will eventually mix different usage scenarios and, hence, is likely to be unsound.

1.1 Overview of the Approach and Contributions

In this paper, the above issues are addressed by proposing a novel process mining algorithm that is able to cope with complex processes by discovering conformant (both sound and complete) models. This is carried out by producing a clear and modular representation of the process, where its relevant variants are explicitly singled out by means of a technique for clustering log traces. Specifically, each variant is modeled by means of a distinct workflow schema $\mathcal{WS}_i$ so that the resulting model for the whole process, called the *disjunctive workflow schema*, is precisely the set of all these individual schemas.

Actually, the greater expressiveness of disjunctive schemas comes with a cost. Indeed, the problem of computing conformant disjunctive schemas and the problem of checking whether a given disjunctive schema at hand is sound *enough* turn out to be intractable.

Therefore, we pragmatically face the process mining problem by means of a greedy strategy based on an iterative, hierarchical refinement of a disjunctive schema, $\mathcal{WS}^\vee$, in which, at each step, a schema $\mathcal{WS}_i \in \mathcal{WS}^\vee$ is selected and the traces supported by $\mathcal{WS}_i$ are partitioned into clusters. Eventually, each cluster is equipped with a refined workflow schema, which is meant to model a specific usage scenario specializing $\mathcal{WS}_i$. Notably, the algorithm is designed in a sophisticated way guaranteeing that each refinement leads to an increasingly sound schema for the given process so that a monotonic search is attained.

Moreover, in order to efficiently partition traces into clusters by means of well-known methods, we investigate an approach for producing a "flat" relational representation of the traces by projecting them onto a set of suitable *features* which are meant to characterize the traces that are not properly modeled by the current workflow schema being refined. Therefore, the paper also accounts for the definition of both an abstract representation for the features and an efficient level-wise algorithm for their computation.

We conclude by noting that all the algorithms above have been implemented, and the results of a thorough experimental activity are also discussed in the paper.

1.2 Organization

The rest of the paper is organized as follows: In Section 2, we report a few preliminaries on workflows, processes, and logs. The problem of discovering a disjunctive process model from some log data is introduced in Section 3, where some relevant computational issues have also been studied. In Section 4, a solution for the basic case where the mined model is nondisjunctive is discussed. Its generalization is reported in Section 5, where the clustering-based approach to process mining and the feature extraction algorithm are illustrated. The two next sections, i.e., Section 6 and Section 7, are devoted to presenting results of the experiments and major related works in the literature, respectively. A few concluding remarks are, finally, sketched in Section 8.

2 Formal Framework: Workflow Schema and Logs

In this section, we introduce the basic notions and notations for formally representing process models that will be used throughout the paper. We point out that a comprehensive formalization of all the facets of workflow modeling is beyond the scope of this paper; the reader interested in expanding on this subject is referred to, e.g., [12], [13], [14], [15], [16], [17]. Specifically, the modeling language is kept simple and less expressive than other existing languages, such as *Petri nets* [18] or *event driven process chains* EPCs [19], [20].

For instance, it does not account for loops, i.e., recursive workflow schemas and iterated executions, which instead can be coped with, e.g., by the WF-net model and the α-algorithm [21], and—as commonly done in the process mining framework—it forces all outgoing and ingoing edges of a given node to have the same type. On the other hand, the model allows for the arbitrary mixing of synchronization and choice constructs, which is sometimes avoided in other formalizations related to process mining.

Moreover, an important feature of our approach is that there is no conceptual limitation in accommodating more sophisticated models and techniques for their manipulation because it is to a large extent independent of the underlying workflow model, which mainly affects the way each cluster of traces is equipped with a schema (cf. algorithm `MineWorkflow` in Section 4). In this respect, the proposed approach is modular and may allow any preexisting process mining algorithm to be used (provided, as we shall see in the following, it satisfies the properties in Theorem 4.3). In fact, the assumptions exploited in the paper were required by the necessity of starting from a simple model enjoying these properties, yet covering important features in workflow specification.

Definition 2.1. *Let P be a process. A* workflow schema $\mathcal{WS}$ *for P is a tuple $\langle A, E, a_0, A_F, \textbf{Fork}, \textbf{Join}\rangle$, where A is a finite set of activities (also called nodes, or tasks), $E \subseteq (A - A_F) \times (A - \{a_0\})$ is an acyclic relation of precedences among activities (whose elements are simply called edges), $a_0 \in A$ is the starting activity, and $A_F \subseteq A$ is the set of final activities. The tuple $\langle A, E, a_0, A_F\rangle$ is often referred to as the* control flow graph *of $\mathcal{WS}$: It states the orderings among the activities involved in P that must be respected during their execution.*

The functions $\textbf{Fork} : (A - A_F) \mapsto \{\text{AND}, \text{OR}, \text{XOR}\}$ and $\textbf{Join} : (A - \{a_0\}) \mapsto \{\text{AND}, \text{OR}\}$, also called local constraints *in the literature, relate the execution of activities that are neighbors of each other in the control flow graph. A node a with $\textbf{Join}(a) = \text{OR}$ (respectively, AND) is an* or-join *(respectively,* and-join*); moreover, a node a with $\textbf{Fork}(a) = \text{OR}$ (respectively, AND, XOR) is an* or-fork *(respectively,* and-fork *and* xor-fork*).*

An example workflow schema is reported in Fig. 1b by means of an intuitive graphical notation. Actually, this is a schema (possibly, the result of some process mining algorithm) for the *OrderManagement* process presented in Section 1.

Each time a workflow is enacted in a workflow management system, it produces an instance that is a suitable subgraph $I = \langle A_I, E_I\rangle$ of the schema satisfying all the constraints. Intuitively, A_I contains those activities that are executed in I and that, therefore, activate their outgoing edges. Notice that the target of these edges is not required to be executed as well, i.e., some branches of the control flow graph may be only partially executed, because a final activity, belonging to another branch, was completed in the meanwhile. Therefore, in the following definition, these kinds of task that are not executed in instance I are explicitly distinguished and included in set A'_I.

Definition 2.2. *Let $\mathcal{WS} = \langle A, E, a_0, A_F, \textbf{Join}, \textbf{Fork}\rangle$ be a workflow schema. Let $I' = \langle A_I \cup A'_I, E'_I\rangle$ be a connected subgraph of $\langle A, E\rangle$, such that the following conditions hold:*

1. $a_0 \in A_I$,
2. $A_I \cap A_F \neq \emptyset$,
3. $\{(a, b) \mid (a, b) \in E'_I \wedge a \in A'_I\} = \emptyset$,
4. *for each* $(a, b) \in E'_I$, $a \in A_I$, *and* $b \in (A'_I \cup A_I)$,
5. *for each* $a \in A_I \cup A'_I - \{a_0\}$,
$$\{b \mid (b, a) \in E'_I \wedge b \in A_I\} \neq \emptyset,$$
6. *for each* $a \in (A_I - \{a_0\})$ *s.t.* $\textbf{Join}(a) = \text{AND}$, $\{(b, a) \mid (b, a) \in E\} \subseteq E'_I$,
7. *for each* $a \in (A_I - A_F)$ *s.t.* $\textbf{Fork}(a) = \text{AND}$, $\{(a, b) \mid (a, b) \in E\} \subseteq E'_I$,
8. *for each* $a \in (A_I - A_F)$ *s.t.* $\textbf{Fork}(a) = \text{XOR}$, $|\{(a, b) \mid (a, b) \in E'_I\}| = 1$, *and*
9. *for each* $a \in (A_I - A_F)$ *s.t.* $\textbf{Fork}(a) = \text{OR}$, $|\{(a, b) \mid (a, b) \in E'_I\}| \geq 1$;

Then, the graph $I = \langle A_I, E'_I \cap (A_I \times A_I)\rangle$ is an instance *of $\mathcal{WS}$ (denoted as $\mathcal{WS} \models I$).*

An example instance for the *OrderManagement* process is reported in Fig. 1c, according to a graphical notation where dashed nodes and edges represent activities and precedences that are activated but not included in the instance. Specifically, the instance models a case where the order is rejected (activity `h` is executed) because the client is not reliable (checked by `f`). However, given that the required product is available in stock, activity `i` is the target of an activated edge, but it is never executed.

We point out that most process-oriented commercial systems only store partial information about process instances by tracing some events related to the execution of its activities. Specifically, the *logs* stored by such systems may range from a simple task sequence (recall, e.g., Fig. 1a) to richer formats (evidencing, e.g., the start and/or the completion of a task). We next describe an abstract representation of a process log which is commonly adopted in the literature.

Let A be the set of task identifiers for the process P; then, a *workflow trace s over A* is a string in A^*, representing a sequence of task executions. Given a trace s, we denote by $s[i]$ the ith task in the corresponding sequence and by $length(s)$ the length of s. The set of all the tasks in s is denoted by $tasks(s) = \bigcup_{1 \leq i \leq length(s)} s[i]$. Finally, a *workflow log for P*, denoted by $\mathcal{L}_P$, is a bag of traces over A, i.e., $\mathcal{L}_P = [\ s \mid s \in A^*]$.

We conclude this section by relating the notion of trace to that of instance.

Definition 2.3. *Let $I = \langle A_I, E_I \rangle$ be an instance of a workflow schema $\mathcal{WS} = \langle A, E, a_0, A_F, \textbf{Join}, \textbf{Fork} \rangle$ for a process P and let s be a trace in $\mathcal{L}_P$. Then, s is* compliant with $\mathcal{WS}$ through I*, denoted by $s \models^I \mathcal{WS}$, if s is a topological sort of I.*

For instance, the trace `abfcgh` is compliant with the instance reported in Fig. 1c, while the traces `afbcgh` and `abfjk` are not.

Notice that the definition above entails that if $s \models^I \mathcal{WS}$, then: $s[1] = a_0$, i.e., the first task in s is the initial activity in $\mathcal{WS}$, $s[length(s)] \in A_F$, i.e., the last task in s is a final activity in $\mathcal{WS}$, and, $tasks(s) = A_I$, i.e., all the tasks in s are executed in I.

We conclude by saying that s is simply *compliant with* $\mathcal{WS}$, denoted by $s \models \mathcal{WS}$, if there exists I such that $s \models^I \mathcal{WS}$. The reader may check, for instance, that the traces in Fig. 1a are compliant with the workflow schema in Fig. 1b as each of them is a topological sort of some instance of that schema.

Before leaving the section, we note that, while logs in real life are often characterized by multiple executions of the same task, our workflow model admits only one execution for each activity in the schema. However, multiple executions can be coped with by means of a syntactic expedient. Let P be a process and let $\mathcal{L}_P$ be a log for it where multiple executions of the same activity are allowed in the same trace. Let a be an activity in P and let $n(\mathrm{a}) \geq 0$ be the maximum number of times it is executed over all the traces in $\mathcal{L}_P$. Then, we can mine a workflow schema $\overline{\mathcal{WS}}$, where each activity a is (virtually) substituted by $n(\mathrm{a})$ distinct activities, say $\mathrm{a}_1, \ldots, \mathrm{a}_{n(\mathrm{a})}$, by exploiting the log $\overline{\mathcal{L}}_P$ derived from $\mathcal{L}_P$ by replacing the ith occurrence of a in any trace by a_i. By construction, $\overline{\mathcal{WS}}$ contains no duplicated activities and, hence, it conforms to Definition 2.1.

3 The Process Model Discovery Problem

In this section, we preliminary introduce and discuss the problem of automatically (re)constructing a workflow schema modeling a given process, on the basis of some log data collected from previous executions. Afterward, a number of variants of the process mining problem are defined and studied from a computational point of view.

3.1 Disjunctive Schemas

Let A be the set of task identifiers for the process P. We assume that the actual workflow schema for P is unknown and we consider the problem of properly identifying it in the set of all the possible workflow schemas having A as the set of activities. To this purpose, we first derive from the log traces an initial workflow schema and we subsequently iteratively refine it into a number of specific schemas, each of them modeling a class of traces having the same *behavioral* characteristics. Therefore, the resulting model is a collection of workflow schemas as defined below.

Definition 3.1. *Let P be a process. A* disjunctive workflow schema *for P, denoted by $\mathcal{WS}^\vee$, is a set $\{\mathcal{WS}^1, \ldots, \mathcal{WS}^m\}$ of workflow schemas for P. The number m is referred to as the* size *of $\mathcal{WS}^\vee$, denoted by $|\mathcal{WS}^\vee|$.*

Any instance I of some $\mathcal{WS}^j \in \mathcal{WS}^\vee$ is also said to be an instance of $\mathcal{WS}^\vee$, denoted by $I \models \mathcal{WS}^\vee$. In fact, the schema is denominated *disjunctive* precisely because the set of its instances is the union of the sets of the instances of each workflow schema in $\mathcal{WS}^\vee$.

Similarly, a trace s is compliant with $\mathcal{WS}^\vee$, denoted by $s \models \mathcal{WS}^\vee$, if s is compliant with some $\mathcal{WS}^j \in \mathcal{WS}^\vee$. Note that, since a trace s does not provide any information about the edges exploited to execute the registered activities, deciding whether s is compliant with the schema $\mathcal{WS}^\vee$ may be, in principle, a complex task. Indeed, while constructing an instance for s, it is crucial that activating edges are properly chosen to satisfy all the local constraints (specifically, the difficulty is with *or-join* tasks activated through *xor-forks*) and to execute all the activities in the trace. The following proposition, whose proof is in [22], makes it clear that a smart strategy can be conceived with this aim so that deciding whether a trace s is compliant with a schema $\mathcal{WS}^\vee$ can be done in polynomial time (in the size of the schema).

Proposition 3.2. *Let $\mathcal{WS}^\vee = \{\mathcal{WS}^1, \ldots, \mathcal{WS}^m\}$ be a disjunctive workflow schema and s be a trace. Let e, n_f, and e_f denote the maximum number of edges, xor-fork nodes, and edges originating in xor-fork nodes, respectively, over all schemas in $\mathcal{WS}^\vee$. Then, deciding whether $s \models \mathcal{WS}^\vee$ is feasible in $O(m \times \max(e, n_f \times e_f))$.*

Let us now assume that a log $\mathcal{L}_P$ for the process P is given. Then, we aim at discovering the disjunctive schema $\mathcal{WS}^\vee$ for P which is as "close" as possible to the actual unknown schema that generated $\mathcal{L}_P$. In the following, the quality of the mined model is evaluated according to two different criteria, namely, the *completeness* and the *soundness*, constraining the discovered model to admiting exactly the traces in the log. Specifically, a (fully) complete workflow is such that all traces in the log at hand are compliant with

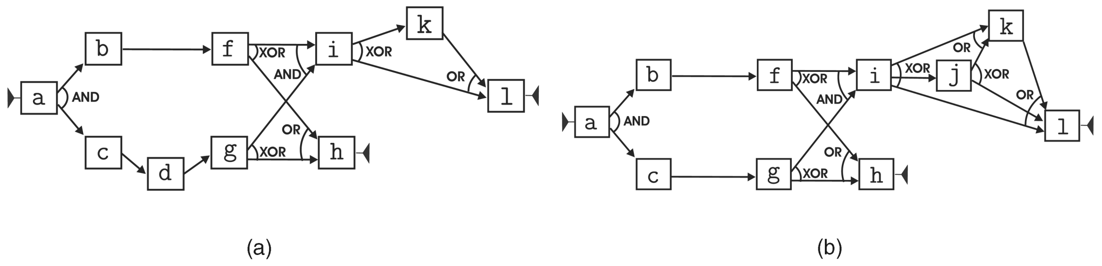


Fig. 2. The two schemas constituting the disjunctive workflow schema $\mathcal{WS}^{\vee}$.

some instance of it, whereas a (fully) sound workflow is such that all of its possible enactments have been actually registered in the log. These two criteria are formalized below.

Definition 3.3. *Let $\mathcal{WS}^{\vee}$ be a disjunctive workflow model and $\mathcal{L}_P$ be a log for the process P. We define:*

- $$soundness(\mathcal{WS}^{\vee}, \mathcal{L}_P) = \frac{|\{s \mid s \in \mathcal{L}_P \ \wedge \ s \models \mathcal{WS}^{\vee}\}|}{|\{s \mid s \models \mathcal{WS}^{\vee}\}|},$$

  i.e., the percentage of traces compliant with $\mathcal{WS}^{\vee}$ that have been registered in the log—the larger the sounder;

- $$completeness(\mathcal{WS}^{\vee}, \mathcal{L}_P) = \frac{|\{s \mid s \in \mathcal{L}_P \ \wedge \ s \models \mathcal{WS}^{\vee}\}|}{|\{s \mid s \in \mathcal{L}_P\}|},$$

  i.e., the percentage of traces in the log that are compliant with $\mathcal{WS}^{\vee}$—the larger the more complete.

Given two real numbers α and β between 0 and 1 (desirable values should be close to 1), we say that $\mathcal{WS}^{\vee}$ is α-sound with regard to $\mathcal{L}_P$, if $soundness(\mathcal{WS}^{\vee}, \mathcal{L}_P) \geq \alpha$; moreover, $\mathcal{WS}^{\vee}$ is β-complete with regard to $\mathcal{L}_P$, if $completeness(\mathcal{WS}^{\vee}, \mathcal{L}_P) \geq \beta$.

We next apply the notions introduced above to a simple explicative example.

Example 3.4. Consider again the workflow schema $\mathcal{WS}_0$ in Fig. 1b. We leave to the careful reader the task of checking that $\mathcal{WS}_0$ admits 20 instances and 276 traces. Let L be the log shown in Fig. 1a. Thus, we have: $soundness(\{\mathcal{WS}_0\}, L) = \frac{16}{276} = 5.797\%$, and

$$completeness(\{\mathcal{WS}_0\}, L) = \frac{16}{16} = 100\%.$$

A different representation for the traces in L is given by the disjunctive workflow schema, which consists of the two workflow schemas shown in Fig. 2. Note that we have: $soundness(\{\mathcal{WS}_1^{\vee}, \mathcal{WS}_2^{\vee}\}, L) = \frac{11}{97} = 11.34\%$ and $completeness(\{\mathcal{WS}_1^{\vee}, \mathcal{WS}_2^{\vee}\}, L) = \frac{11}{16} = 68.75\%$. Indeed, $\{\mathcal{WS}_1^{\vee}, \mathcal{WS}_2^{\vee}\}$ models 97 distinct instances, 64 through $\mathcal{WS}_1^{\vee}$ and 33 through $\mathcal{WS}_2^{\vee}$. However, the completeness value is lower than 1 as some traces in L (namely, $s_8, \ldots, s_{12}$) are not compliant with $\{\mathcal{WS}_1^{\vee}, \mathcal{WS}_2^{\vee}\}$. Interestingly, this example shows how the soundness of a given schema can be increased by replacing it with a set of more specific schemas. This strategy is exploited in the mining algorithm presented in Section 5, which also prevents completeness losses.

Since soundness and completeness are the parameters to be taken into account while mining process models, it is relevant to study their "intrinsic" complexity. As proven next, checking soundness is a hard task, surprisingly even in the case where the model is given at hand and does not need to be discovered. The result evidences that our mining problem is structurally harder than traditional learning problems, where the functions to be optimized can be checked in polynomial time for any candidate solution.

Proposition 3.5. *Let $\mathcal{WS}^{\vee}$ be a disjunctive workflow model, and $\mathcal{L}_P$ be a log for a process P. Then, deciding whether $\mathcal{WS}^{\vee}$ is 1-*complete *with regard to $\mathcal{L}_P$ is feasible in time $O(|\{s \mid s \in \mathcal{L}_P\}| \times m \times \max(e, n_f \times e_f))$,*[1] *with one scan of $\mathcal{L}_P$ only. Moreover, deciding whether $\mathcal{WS}^{\vee}$ is 1-*sound *with regard to $\mathcal{L}_P$ is* co-NP-*complete. Hardness holds even for $|\mathcal{WS}^{\vee}| = 1$.*

Proof. To check 1-completeness, we can repeat the procedure in Proposition 3.2, for each trace s in $\mathcal{L}_P$; therefore, the result easily follows.

Let us now prove that checking for 1-soundness is co-NP-complete. Consider the complementary problem, say $\overline{soundness}$, of deciding whether there is a trace s such that $s \models \mathcal{WS}^{\vee}$ and $s \notin \mathcal{L}_P$. Problem $\overline{soundness}$ is obviously in NP; we next show that it is NP-complete. To this purpose, we recall that, in [23], it has been proven that deciding whether a given workflow schema $\mathcal{WS}$ admits an instance is NP-complete. We construct the following instance of the problem $\overline{soundness}$: $\mathcal{WS}^{\vee} = \{\mathcal{WS}\}$ and $\mathcal{L}_P = \emptyset$. Then, the answer of $\overline{soundness}$ is "yes" if and only if $\mathcal{WS}$ admits an instance; hence $\overline{soundness}$ is NP-complete as well. It turns out that deciding 1-soundness is co-NP-complete. □

3.2 Exact and Maximum Process Discovery

Armed with the framework outlined so far, we are in the position of formalizing the problem we want to deal with. Basically, we aim at discovering a disjunctive schema $\mathcal{WS}^{\vee}$ for a given process P which is α-*sound* and β-*complete*, for some given α and β. However, it is easy to see that a trivial schema satisfying the above conditions always exists (even for $\alpha = 1$ and $\beta = 1$), consisting of the union of exactly one workflow modeling each distinct trace in $\mathcal{L}_P$. Unfortunately,

1. See the notation in Proposition 3.2.

such a model would be an overly detailed and complex representation of the process P, with its size being $|\mathcal{WS}^\vee| = |\mathcal{L}_P| = |\{s \mid s \in \mathcal{L}_P\}|$. We therefore introduce a bound on the number of schemas in $\mathcal{WS}^\vee$ for defining the basic problem we shall study in the paper.

Let $\mathcal{L}_P$ be a workflow log for the process P. Given two real numbers α and β, and a natural number $m > 0$, the *Exact Process Discovery problem*, denoted by $\mathrm{EPD}(\mathcal{L}_P, \alpha, \beta, m)$, consists of finding (if any) an α-sound and β-complete disjunctive workflow schema $\mathcal{WS}^\vee$, such that $|\mathcal{WS}^\vee| \leq m$. Actually, we can show (see [22]) that the Exact Process Discovery problem can be solved in polynomial time only for trivial cases (unless $\mathrm{P} = \mathrm{NP}$).

Theorem 3.6. *Given a log $\mathcal{L}_P$ for a process P and a natural number $m > 0$,* $\mathrm{EPD}(\mathcal{L}_P, 1, 1, m)$ *always admits a trivial solution if $|\mathcal{L}_P| \leq m$. Otherwise, i.e., if $|\mathcal{L}_P| > m$, deciding whether* $\mathrm{EPD}(\mathcal{L}_P, 1, 1, m)$ *admits a solution is in Σ_2^{P} and* NP-hard.

Since it could happen that EPD does not have solutions (cf. Theorem 3.6) and since it is difficult to check whether we are in such a situation (cf. Proposition 3.5), we restate the process discovery problem in such a way that it always admits a solution. Specifically, we allow the sacrifice of enough portions of soundness to get a result; however, we impose a strict bound on the completeness (which is usually 1) since modeling all the traces in the log is often an important requirement for the mined schema.

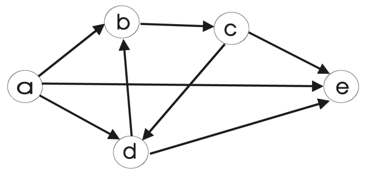
Definition 3.7. *Let $\mathcal{L}_P$ be a workflow log for the process P. Given a natural number m, the* Maximum Process Discovery problem, *denoted by* $\mathrm{MPD}(\mathcal{L}_P, m)$, *consists of finding a 1-complete disjunctive workflow schema $\mathcal{WS}^\vee$ such that $|\mathcal{WS}^\vee| \leq m$ and $soundness(\mathcal{WS}^\vee, \mathcal{L}_P)$ is the maximum over all the 1-complete schemas.*

However, the problem MPD is also untractable by a straightforward reduction from EPD. Moreover, we suspect a hardness result for the second level of the polynomial hierarchy. Therefore, it comes as no surprise that we pragmatically face the problem by using a heuristic approach in which we mine a preliminary, possibly not sound enough, model, and iteratively refine it. Specifically, the approach is designed in a way that each time a refinement is made, the model is guaranteed to increase in soundness. The careful reader at this point may understand the importance of this property since it allows a monotonic search in the space of the solutions by never checking for the soundness of the current model, which is unfeasible.

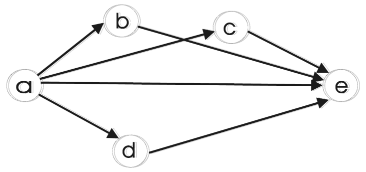
4 Mining a Workflow Schema

In this section, we start the description of the algorithm for $\mathrm{MPD}(\mathcal{L}_P, m)$ by proposing a solution for the case $m = 1$. Our approach is simple, but it enjoys some nice properties that are crucial for the extension to the case $m > 1$ and which have been not fully considered while designing previous approaches in the literature.

Throughout this section, we assume that a log $\mathcal{L}_P$ for the process P over tasks A is given. For the sake of exposition, we assume that each trace contains the initial activity a_0 as the first task. We start by introducing notions for expressing precedence relationships between activities, which are derived from occurrences in the logs:



(a)



(b)

Fig. 3. Example 4.1: (a) Dependency graph and (b) control flow graph.

- The *dependency graph* for $\mathcal{L}_P$ is the graph $D_{\mathcal{L}_P} = \langle A, E \rangle$, where

$$E = \{(a, b) \mid \exists s \in \mathcal{L}_P,\ i \in \{1, \ldots, length(s) - 1\} \text{ s.t. } a = s[i] \wedge b = s[i+1]\}.$$

- Two activities a and b in A are *parallel* in $\mathcal{L}_P$, if they occur in some cycle of $D_{\mathcal{L}_P}$.
- Given two activities a and b in A, we say that a *precedes* b in $\mathcal{L}_P$, denoted by $a \rightarrow b$, if a and b are not parallel, and there is a path from a to b in $D_{\mathcal{L}_P}$.

Example 4.1. Consider the log $L = \{\mathrm{abcde}, \mathrm{adbce}, \mathrm{ae}\}$, concerning a process P. The dependency graph for L is shown in Fig. 3a. Notice, for instance, that `a`, `b`, and `c` are parallel activities in L, and that `a` $\rightarrow$ `c` and `b` $\rightarrow$ `e` hold.

Armed with these notions, we can now describe the algorithm `MineWorkflow`, which is shown in Fig. 4. The algorithm starts by constructing the dependency graph, which is subsequently modified in Steps 2-11 in order to remove the cycles, which, in fact, correspond to sets of parallel activities. Actually, removing an edge is a critical step since it must be carried out by preserving connectivity. Therefore, each time an edge, say (a, b), is removed from E, we preserve the paths by connecting a and b with some preceding activity (Step 5) and with some following one (Step 8) for each of the traces in $\mathcal{L}_P$.

Notice that there are several alternative graphs that may support all the logs at hand; the algorithm has been designed for introducing as few spurious traces as possible so that soundness is heuristically maximized. For instance, a schema with a_0 connected to all the other activities will be a complete schema as well, but it is likely to be terribly unsound. Our idea is instead to preserve the paths in the flow (ideally, in the absence of cycles, we can get even

```
Input: A log $\mathcal{L}_P$.
Output: A workflow schema $\mathcal{WS} = \langle A, E, a_0, A_F, \mathit{Join}, \mathit{Fork}\rangle$.
Method: Perform the following steps:
1   $\langle A, E\rangle := D_{\mathcal{L}_P}$;    //nodes and edges are initially those of the dependency graph
2   for each $(a,b) \in E$ s.t. $a$ and $b$ are parallel in $\mathcal{L}_P$ do    //remove cycles
3     $E := E - \{(a,b)\}$;
4     for each $s \in \mathcal{L}_P$ s.t. $\{a,b\} \subseteq tasks(s)$ do    //update edges
5       $pre := s[i]$, where $s[i] \rightarrow a \wedge s[i] \rightarrow b$ and not exists $s[k]$ with $k > i$ s.t. $s[k] \rightarrow a \wedge s[k] \rightarrow b$;
6       $E := E \cup \{(pre, a)\} \cup \{(pre, b)\}$;
7       $post := s[j]$, where $a \rightarrow s[j] \wedge b \rightarrow s[j]$ and not exists $s[h]$ with $h < j$ s.t. $a \rightarrow s[h] \wedge b \rightarrow s[h]$;
8       $E := E \cup \{(a, post)\} \cup \{(b, post)\}$;
9     end for
10  end for
11  $a_0 := s[1]$, no matter of which trace $s \in \mathcal{L}_P$ is selected;   $A_F := \{a \in A \mid \nexists b \in A \text{ s.t. } a \rightarrow b\}$;
12  for each $a \in A$ do    //construction of local constraints
13    if $\forall s \in \mathcal{L}_P$ s.t. $a \in tasks(s)$, it holds that $\forall c$ s.t. $(a,c) \in E$, $c \in tasks(s)$ then $\mathit{Fork}(a) = \mathtt{AND}$;
14    else if $\forall s \in \mathcal{L}_P$ s.t. $a \in tasks(s)$, $|\{c \mid (a,c) \in E \wedge c \in tasks(s)\}| = 1$ then $\mathit{Fork}(a) = \mathtt{XOR}$;
15    else $\mathit{Fork}(a) = \mathtt{OR}$;
16    if $\forall s \in \mathcal{L}_P$ s.t. $a \in tasks(s)$, $(c,a) \in E \Rightarrow c \in tasks(s)$ then $\mathit{Join}(a) = \mathtt{AND}$;
17    else $\mathit{Join}(a) = \mathtt{OR}$;
18  end for
19  return $\langle A, E, a_0, A_F, \mathit{Join}, \mathit{Fork}\rangle$;
```

Fig. 4. **Algorithm** `MineWorkflow`: A solution for $\mathtt{MPD}(\mathcal{L}_P, 1)$.

1-sound schemas). Accordingly, in Steps 12-18, the algorithm builds the local constraints by applying the most stringent possible choices for each activity. Specifically, notice that or-forks and or-joins are likely to deteriorate the soundness and should be therefore avoided if possible.

Example 4.2. Let us turn back to Example 4.1, and apply the `MineWorkflow` algorithm. In Steps 2-10, the edges (`b`, `c`), (`c`, `d`), and (`d`, `b`) are removed from the graph in Fig. 3a, since they are parallel in L. The connectivity of the graph is reestablished by connecting `a` to `c`, and `b` to `e`; actually, other edges are processed that were already in the graph. In Step 11, `a` becomes the starting activity and `e` the only final one. The resulting control flow is shown in Fig. 3b. Finally, the associated constraints are such that: ***Fork*** and ***Join*** both assign the value `AND` to each of the activities `b`, `c`, and `d`, for they having just one predecessor and one successor. Moreover, the initial activity `a` (respectively, the final activity `e`) is associated with an `OR` value by the ***Fork*** (respectively, ***Join***) function, since the trace `ae` in the log contains only one of the successors of `a` (respectively, predecessors of `e`).

We can now conclude the description of the algorithm by stating a number of relevant properties. First, it is not difficult to see that the mined schema is 1-complete; more interestingly, we are able to show (see [22]) that it satisfies a kind of "monotonicity" property that is crucial for guaranteeing the correctness of the greedy strategy we shall exploit in the following section for the general problem $\mathtt{MPD}(\mathcal{L}_P, m)$, with $m > 1$.

Theorem 4.3. *The* `MineWorkflow` *algorithm on input $\mathcal{L}_P$ computes a workflow schema $\mathcal{WS}$ in linear time in the size of $\mathcal{L}_P$ and satisfies the following conditions:*

- ***Maximum Completeness Condition****: $\mathcal{WS}$ is a 1-complete workflow schema with regards to $\mathcal{L}_P$*
- ***Monotonicity Condition****: Let $\mathcal{L}'_P$ be a log such that $\mathcal{L}'_P \subseteq \mathcal{L}_P$ and let $\mathcal{WS}'$ be the output of* `MineWorkflow` *on input $\mathcal{L}'_P$. Then, the set of traces compliant with $\mathcal{WS}'$ is a subset of the set of traces compliant with $\mathcal{WS}$, i.e., $\{s \mid s \models \mathcal{WS}'\} \subseteq \{s \mid s \models \mathcal{WS}\}$.*

As a further remark, note that the proposed algorithm can be extended to cope with noise by slightly modifying the construction of the dependency graph. Specifically, we can use a threshold ϱ such that an edge (a,b) is in a suitable dependency graph, say $D^{\varrho}_{\mathcal{L}_P}$, if and only if a occurs before b in at least $\varrho \times |\mathcal{L}_P|$ traces. Clearly enough, this construction avoids the introduction of spurious dependencies that are unfrequent in $\mathcal{L}_P$.

5 Clustering Workflow Traces

In order to solve the $\mathtt{MPD}(\mathcal{L}_P, m)$ problem, we exploit the idea of iteratively and incrementally refining a schema, starting with a preliminary nondisjunctive model which can be mined by the algorithm `MineWorkflow`. In a nutshell, we propose a greedy solution, implemented in the algorithm `ProcessDiscover`, that computes the mined schema $\mathcal{WS}^{\vee}$ through a hierarchical clustering in which the current disjunctive schema (equipped with a subset of the

```
Input: Problem MPD($\mathcal{L}_P$,$m$), natural numbers $maxF$, $k$ and $\ell$, thresholds $\sigma$ and $\gamma$.
Output: A process model.
Method: Perform the following steps:
  1   $\mathcal{WS}_0^1$ :=MineWorkflow($\mathcal{L}_P$);    //See Sec. IV
  2   $\mathcal{WS}^\vee := \{\mathcal{WS}_0^1\}$;  $\mathcal{L}(\mathcal{WS}_0^1) := \mathcal{L}_P$;  $noRefinement$ = false;
  3   while $|\mathcal{WS}^\vee| < m$ or $noRefinement$ = true do
  4     $\mathcal{WS}_i^j$ :=$selectSchema(\mathcal{WS}^\vee)$;
  5     $oldSize := |\mathcal{WS}^\vee|$;
  6     $refineWorkflow(i,j,k,\sigma,\gamma,\ell,maxF)$    //see Thm. V.1
  7     if $oldSize = |\mathcal{WS}^\vee|$ then $noRefinement$ = true;
  8   end while
  9   return $\mathcal{WS}^\vee$;        //see Thm. V.1

Procedure refineWorkflow($i$: refinement, $j$: schema, $k$: branching factor, $\sigma$,$\gamma$: thresholds, $\ell$,$maxF$: natural numbers)
  P1    $\mathcal{F}$ :=FindFeatures($\mathcal{L}(\mathcal{WS}_i^j)$,$\mathcal{WS}_i^j$,$\sigma$,$\gamma$,$\ell$,$maxF$);
  P2    if $|\mathcal{F}| \geq 1$ then
  P3      $\mathcal{R}(\mathcal{WS}_i^j)$ :=Project($\mathcal{L}(\mathcal{WS}_i^j), \mathcal{F}$);    //see Sec. V-B
  P4      $h := \max\{h \mid \mathcal{WS}_{i+1}^h \in \mathcal{WS}^\vee\}$;
  P5      $\langle \mathcal{L}(\mathcal{WS}_{i+1}^{h+1}), ..., \mathcal{L}(\mathcal{WS}_{i+1}^{h+k}) \rangle := k\text{-}means(\mathcal{R}(\mathcal{WS}_i^j))$;
  P6      $\mathcal{WS}^\vee := \mathcal{WS}^\vee - \{\mathcal{WS}_i^j\}$;
  P7      for each $\mathcal{WS}_{i+1}^h$ do
  P8        $\mathcal{WS}_{i+1}^h$ :=MineWorkflow($\mathcal{L}(\mathcal{WS}_{i+1}^h)$);    //see Sec. IV
  P9        $\mathcal{WS}^\vee = \mathcal{WS}^\vee \cup \{\mathcal{WS}_{i+1}^h\}$;
  P10     end for
  P11   end if;
```

Fig. 5. **Algorithm** `ProcessDiscover`.

input log) is stepwise refined into k clusters of traces modeled by possibly different schemas, with the aim of increasing its soundness yet preserving maximum completeness. In this section, we discuss the details of such an algorithm.

5.1 Algorithm `ProcessDiscovery`

The algorithm `ProcessDiscover` is shown in Fig. 5. It first mines, in Step 1, a workflow schema $\mathcal{WS}_0^1$, by means of the procedure `MineWorkflow` described in Section 4. This workflow is associated with all the traces in $\mathcal{L}_P$ and becomes the starting point of the computation; indeed, the disjunctive workflow schema $\mathcal{WS}^\vee$ initially contains $\mathcal{WS}_0^1$ only (see Step 2). Then, the algorithm starts refining the schema. In particular, each workflow schema, say $\mathcal{WS}_i^j$, eventually inserted in $\mathcal{WS}^\vee$, is identified by the number i of refinements occurred since the beginning of the computation and by an index j for distinguishing the schemas at the same refinement level. Each schema $\mathcal{WS}_i^j$ is also equipped with a set of traces that $\mathcal{WS}_i^j$ is able to model, denoted by $\mathcal{L}(\mathcal{WS}_i^j)$. These sets of traces can be viewed as clusters of the original log.

At each step, `ProcessDiscover` selects a schema $\mathcal{WS}_i^j \in \mathcal{WS}^\vee$ for being refined (Step 4) by exploiting the function *refineWorkflow*. To this purpose, the most natural strategy is to select the schema having the minimum value of soundness over all the schemas in $\mathcal{WS}^\vee$. In order to get an efficient implementation, we pragmatically suggest exploiting an approximation of this approach, where the schema having the maximum number of or-forks is chosen—these nodes introduce, in fact, nondeterminism and possibly spurious traces. Notably, in our current implementation (see Section 6), the user is allowed to interactively select the cluster to refine at each step so that any arbitrary, application-dependent strategy may be adopted.

The refinement is carried out by "partitioning" the traces associated with $\mathcal{WS}_i^j$ in a way that guarantees the resulting schema to increase in soundness. Actually, in order to reuse classical clustering method and, specifically in our implementation, the *k-means* algorithm, the procedure *refineWorkflow* translates the log $\mathcal{L}(\mathcal{WS}_i^j)$ into flat relational data, denoted by $\mathcal{R}(\mathcal{WS}_i^j)$, by means of the procedures `FindFeatures` and `Project`, which will be discussed in Section 5.2. The basic idea is to identify a set of relevant features that are assumed to characterize the traces in the cluster, thereby leading to viewing each trace as a Boolean tuple over the space of such features. In particular, if more than one feature is identified, it computes the clusters $\mathcal{WS}_{i+1}^{h+1}, \ldots, \mathcal{WS}_{i+1}^{h+k}$, where h 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 add them to the disjunctive schema $\mathcal{WS}^\vee$.

Finally, for each schema added to $\mathcal{WS}^\vee$, the procedure `MineWorkflow` described in Section 5 is applied, so that the control flow and the local constraints are mined as well.

Note that a main point of the algorithm is fixing the number k of new schemas 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 would then expect that the latter case is the most desirable. This is not necessarily true; rather, there are three basic reasons for preferring a hierarchical computation, with several refinements:

1. Exploiting a large value of k would be beneficial only if it is possible to encode traces in a metric space by guaranteeing that standard clustering algorithms produce a solution maximizing the soundness; recall, indeed, that our basic aim is to identify bunches of execution traces which can be soundly modeled by a workflow schema. However, given that the notion of soundness is computationally intractable (cf. Proposition 3.5) and that the MPD problem is likely to be hard for the second level of the polynomial hierarchy (cf. Theorem 3.6), we believe that this cannot be done efficiently. To see this from another perspective, we can say that, differently from traditional clustering problems where the compactness of the clusters is measured according to the same metric used for evaluating the similarity between pairs of entities (e.g., Euclidean distance in a metric space), in the MPD problem the objective function (soundness) cannot be related in a straightforward manner with some kind of likeness among traces. An iterative approach based on stepwise refinements guaranteeing that each refinement leads to an increase in soundness is quite an effective solution to this problem. Moreover, it allows a monotonic search in the space of the solutions by never checking for the soundness of the current model.
2. The result of the hierarchical clustering is a taxonomy of workflow schemas whose leaves encode, in fact, the mined disjunctive model (recall that each workflow is in fact equipped with a level i and is the result of a refinement of some workflow with level $i-1$). Such a tree-based representation is relevant because it gives more insights on the properties of the modeled workflow instances and provides an intuitive and expressive description of the process behavior at different levels of detail. The exploitation of this structure for semantic knowledge consolidation tasks has been recently discussed in [24].
3. Disjunctive models can be used not only to get an effective comprehension of complex processes, but also as executable models supporting further coming instances of the process. Indeed, as soon as there is a new enactment, the most appropriate mined variant of the process should be selected based on the environment (e.g., users and data values). Clearly enough, shifting the choice of the variant to the very first step of the enactment may be undesirable in several situations (see, e.g., studies on branching bisimulation [25], [26]). In fact, an interesting strategy to support the enactment is to exploit the hierarchical structure of the mined model. At the very beginning, the most general model (i.e., the root of the hierarchy) may be selected for being enacted. Then, as soon as a choice is made which allows univocal determination of the specific variant (of the current schema) being actually executed, the hierarchy can be traversed, and one child of the current node can be selected to be the actual workflow schema. The technique can be iterated till a leaf of the hierarchy is selected. Therefore, the more the schema has a tree-like structure, the more we are free to decide the actual moment in which a specific schema has to be associated with the current enactment.

We can now conclude the description of the algorithm `ProcessDiscover` by stating its main properties. The careful reader will notice that the properties of `MineWorkflow` are in fact crucial now.

Theorem 5.1. *Let $\mathcal{WS}^{\vee}$ be the output of* `ProcessDiscover` *applied on input* $\mathrm{MPD}(\mathcal{L}_P, m)$*. Then, 1) $\mathcal{WS}^{\vee}$ is 1-complete with regards to $\mathcal{L}_P$. 2) Let $\mathcal{WS}^{\vee}_b$ be an α-sound disjunctive schema, and let $\mathcal{WS}^{\vee}_a$ be the α'-sound disjunctive schema obtained updating $\mathcal{WS}^{\vee}_b$ by means of the invocation of refineWorkflow in Step 6. Then, $\alpha' \geq \alpha$. 3) The main loop (Steps 3-8) is repeated m times at most.*

Proof.

1. Assume that the output $\mathcal{WS}^{\vee}$ of `ProcessDiscover` on input $\mathrm{MPD}(\mathcal{L}_P, m)$ is of the form $\{\mathcal{WS}_1, \ldots, \mathcal{WS}_n\}$. Recall, preliminarily, that each schema $\mathcal{WS}_i$ is also equipped with a subset of $\mathcal{L}_P$, denoted by $\mathcal{L}(\mathcal{WS}_i)$. It is easy to see that the logs $\mathcal{L}(\mathcal{WS}_1), \ldots, \mathcal{L}(\mathcal{WS}_n)$ form, in fact, a partition of $\mathcal{L}_P$. Indeed, the traces associated with $\mathcal{WS}^1_0$ coincide with those in $\mathcal{L}_P$ (see Step 2); moreover, each time a schema is refined, its associated traces are simply clustered in Step P5, so that the property of being a partition of $\mathcal{L}_P$ is preserved after any invocation of the *refineWorkflow* procedure.

   Finally, the result follows because of the 1-completeness (by the property of `MineWorkflow` in Step 1 and Step P8) of each schema $\mathcal{WS}_i \in \{\mathcal{WS}_1, \ldots, \mathcal{WS}_n\}$.
2. Let $\mathcal{WS}^{\vee}_b$ be an α-sound disjunctive schema and let $\mathcal{WS}^{\vee}_a$ be the α'-sound disjunctive schema obtained updating $\mathcal{WS}^{\vee}_b$ by means of the invocation of *refineWorkflow*. Let us preliminary recall that:

$$\alpha = \frac{|\{s \mid s \in \mathcal{L}_P \wedge s \models \mathcal{WS}^{\vee}_b\}|}{|\{s \mid s \models \mathcal{WS}^{\vee}_b\}|};$$

$$\alpha' = \frac{|\{s \mid s \in \mathcal{L}_P \wedge s \models \mathcal{WS}^{\vee}_a\}|}{|\{s \mid s \models \mathcal{WS}^{\vee}_a\}|}.$$

   Then, by property 1 above, both $\mathcal{WS}^{\vee}_b$ and $\mathcal{WS}^{\vee}_a$ are 1-complete. Therefore,

$$\frac{|\{s \mid s \in \mathcal{L}_P \wedge s \models \mathcal{WS}^{\vee}_b\}|}{|\{s \mid s \in \mathcal{L}_P\}|} = \frac{|\{s \mid s \in \mathcal{L}_P \wedge s \models \mathcal{WS}^{\vee}_a\}|}{|\{s \mid s \in \mathcal{L}_P\}|} = 1.$$

Assume now, without loss of generality, that *refineWorkflow* finds at least two features; otherwise, $\mathcal{WS}_b^\vee = \mathcal{WS}_a^\vee$ holds. Then, let $\mathcal{WS}_b^\vee = \{\mathcal{WS}_1, \ldots, \mathcal{WS}_n\}$ and let $\mathcal{WS}_n$ be the schema refined by *refineWorkflow* so that

$$\mathcal{WS}_a^\vee = \{\mathcal{WS}_1, \ldots, \mathcal{WS}_{n-1}, \mathcal{WS}_{n+1}, \ldots \mathcal{WS}_{n+k}\}$$

is the disjunctive schema after which the refinement of WS_n is performed—notice that we are assuming, without loss of generality, that $\mathcal{WS}_n$ is the schema chosen for being refined. Now, observe that

$$\{s \mid s \models \mathcal{WS}_b^\vee\} = \bigcup_{i=1..n} \{s \mid s \models \mathcal{WS}_i\},$$

and

$$\{s \mid s \models \mathcal{WS}_a^\vee\} = \bigcup_{i=1..n-1,n+1..n+k} \{s \mid s \models \mathcal{WS}_i\}.$$

Recall that each schema of the form $\mathcal{WS}_i$, with $n+1 \leq i \leq n+k$, is obtained by mining $\mathcal{L}(\mathcal{WS}_i)$ by means of `MineWorkflow` and that $\{\mathcal{L}(\mathcal{WS}_{n+1}), \ldots, \mathcal{L}(\mathcal{WS}_{n+k})\}$ is a partition of $\mathcal{L}(\mathcal{WS}_n)$ by result of clustering in Step P5. It follows that $\mathcal{L}(\mathcal{WS}_i) \subseteq \mathcal{L}(\mathcal{WS}_n)$, for each index i with $n+1 \leq i \leq n+k$. Then, since `MineWorkflow` satisfies the monotonicity condition (cf. Theorem 4.3), the set $\{s \mid s \models \mathcal{WS}^i\}$ is contained in the set $\{s \mid s \models \mathcal{WS}^n\}$. Clearly, this entails that $\{s \mid s \models \mathcal{WS}_a^\vee\} \subseteq \{s \mid s \models \mathcal{WS}_b^\vee\}$ and, consequently, $|\{s \mid s \models \mathcal{WS}_a^\vee\}| \leq |\{s \mid s \models \mathcal{WS}_b^\vee\}|$. Thus, $\frac{\alpha'}{\alpha} \geq 1$ holds as well.

3. Termination in m iterations at most is guaranteed by the check in Step 3. □

5.2 Dealing with Relevant Features

The last aspect to be analyzed in the algorithm `Process Discovery` is the way the procedures `FindFeatures` and `Project` are carried out. Roughly speaking, the former identifies a set $\mathcal{F}$ of relevant features, whereas the latter projects the traces into a vectorial space whose components correspond to these features. This is done with the aim of reducing the problem of clustering workflow traces to a standard clustering problem for which efficient algorithms have already been proposed in literature.

Actually, the idea of representing the data set at hand by using a proper set of features has been exploited for efficiently handling data mining problems (see, e.g., [27], [28], [29], [30], [31]); as an instance, in [30], the problem of classifying sequences is dealt with by considering frequent subsequences of them as relevant features.

Clearly enough, changing the domain of interest dramatically affects the notion of relevant feature, which is strongly application dependent. Specifically, in the case of workflow executions, the identification of relevant features is aimed at having clusters which can be modeled by means of sound schemas. Generally, a schema with a low value of soundness is such that it mixes different execution scenarios that cannot be kept separate by means of dependencies in the control flow and local constraints only. Thus, a simple way of increasing soundness is precisely to single out that kind of (frequent) "behavior" that is not properly modeled by the workflow schema. This is explained below.

Example 5.2. Consider again the sample workflow schema and the associated log reported in Fig. 1. Let us try to identify some relevant features to be used for the clustering. Consider, for instance, the sequences `abefi` and `ik`, and notice that they frequently occur in the log (five times each). However, their combination, i.e., `abefik`, never occurs in the log, which is unexpected by looking at the control flow only. Similar considerations also apply when noticing the absence in the log of the sequence `acdgij`, which is strange due to frequent occurrence of both the sequential patterns `acdgi` and `ij`.

Intuitively, the situations above witness some kinds of behavioral constraint. For instance, it may be the case that a fidelity discount is not applied for new clients and that the mail department can be contacted only when it was not necessary to check the availability of external suppliers. As a result of this behavior, since both `abefik` and `acdgij` are expected but not in the log, the workflow schema is likely to be unsound. The notion of relevant feature reported below is aimed at capturing such scenarios.

Let $\mathcal{L}_P$ be a log, $\mathcal{WS} = \langle A, E, a_0, A_F, \textbf{\textit{Fork}}, \textbf{\textit{Join}}\rangle$ be a workflow schema, and σ be a threshold, i.e., a real number such that $0 \leq \sigma \leq 1$. Then, we say that a sequence $\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_\mathrm{h}$ of activities (i.e., a string in A^*) is σ-*frequent* in $\mathcal{L}_P$ if, for each pair of consecutive activities $(\mathtt{a}_i, \mathtt{a}_{i+1})$, with $1 \leq i \leq h-1$, there is a path from $\mathtt{a}_i$ to $\mathtt{a}_{i+1}$ in $\langle A, E\rangle$, and if

$$|\{s \in \mathcal{L}_P \mid \mathtt{a}_1 = s[i_1], \ldots, \mathtt{a}_h = s[i_h] \wedge i_1 < \ldots < i_h\}|/|\mathcal{L}_P| > \sigma.$$

Definition 5.3. *A* discriminant rule *(or* feature*) with* threshold $\langle\sigma,\gamma\rangle$ *in a log* $\mathcal{L}_P$ *is an expression* ϕ *of the form* $[\mathtt{a}_1\ldots\mathtt{a}_\mathrm{h}] - \not\rightarrow_{\langle\sigma,\gamma\rangle} \mathtt{a}$ *such that: 1)* $\mathtt{a}_1\ldots\mathtt{a}_\mathrm{h}$ *is* σ-frequent *in* $\mathcal{L}_P$*, 2)* $\mathtt{a}_\mathrm{h}\mathtt{a}$ *is* σ-frequent *in* $\mathcal{L}_P$*, and 3)* $\mathtt{a}_1\ldots\mathtt{a}_\mathrm{h}\mathtt{a}$ *is not* γ-frequent *in* $\mathcal{L}_P$*.*

Notice that, in the above definition, we considered a second threshold (γ) for evaluating whether the resulting string is frequent: The lower γ is, the more unexpected the rule is. In the extreme case, i.e., $\gamma = 0$, the workflow mixes two scenarios that are completely independent of each other. This is the case, for instance, of the schema and the log in Fig. 1, where both $[\mathtt{abefi}] - \not\rightarrow_{\langle\sigma,0\rangle} \mathtt{k}$ and $[\mathtt{acdgi}] - / \rightarrow_{\langle\sigma,0\rangle} \mathtt{j}$ are discriminant rules for $\sigma < 5/16$.

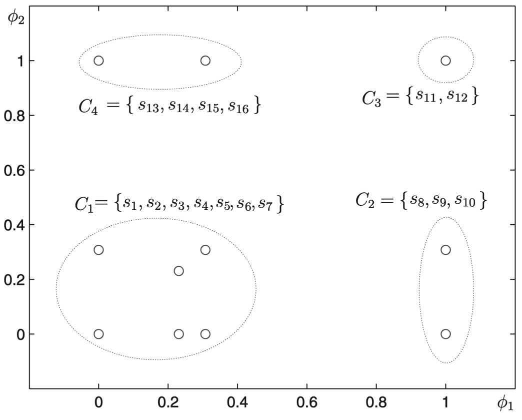
Actually, while discovering discriminant rules, we are interested in those satisfying some additional minimality requirements.

Definition 5.4. *A feature* $\phi : [\mathtt{a}_1\ldots\mathtt{a}_\mathrm{h}] - \not\rightarrow_{\langle\sigma,\gamma\rangle} \mathtt{a}$ *is* minimal *if the following conditions (aimed at avoiding redundancies) are satisfied:*

- *does not exist* $\mathtt{b}$ *with* $[\mathtt{a}_1\ldots\mathtt{a}_\mathrm{h}] - \not\rightarrow_{\langle\sigma,\gamma\rangle} \mathtt{b}$*, such that* $\mathtt{ab}$ *is* σ-frequent *in* $\mathcal{L}_P$*, and*
- *does not exist* $[\mathtt{c}_1\ldots\mathtt{c}_\mathrm{k}] - \not\rightarrow_{\langle\sigma,\gamma\rangle} \mathtt{a}$*, such that* $tasks(\mathtt{c}_1\ldots\mathtt{c}_\mathrm{k}) \subseteq tasks(\mathtt{a}_1\ldots\mathtt{a}_\mathrm{h})$*.*

	ϕ_1	ϕ_2
s_1	0,23	0,23
s_2	0,23	0,23
s_3	0,23	0
s_4	0,30	0,30
s_5	0	0,30
s_6	0,30	0
s_7	0	0
s_8	1	0,30
s_9	1	0,30
s_{10}	1	0
s_{11}	1	1
s_{12}	1	1
s_{13}	0	1
s_{14}	0	1
s_{15}	0	1
s_{16}	0,30	1

(a)



(b)

Fig. 6. Sample traces: (a) Projection in the feature space and (b) clusters.

Once minimal features have been discovered, the procedure `Project` maps each trace s in the log $\mathcal{L}_P$ into a point of a suitable vectorial space where the *k-means* algorithm can operate. The vectorial space has as many dimensions as the number of features considered and each trace s is mapped to the point $\overrightarrow{s}$ as follows. Let $\phi : [\mathtt{a}_1 \ldots \mathtt{a}_h] - \not\rightarrow_{\langle\sigma,\gamma\rangle} \mathtt{a}$ be a feature computed by `FindFeatures`, then the value of the component of $\overrightarrow{s}$ associated with ϕ is: 0, if $\mathtt{a} \in tasks(s)$, or

$$\frac{\sum_i |\{s[i]\} \cap \{\mathtt{a}_1, \ldots, \mathtt{a}_h\}| \times length(s)^{length(s)-i}}{\sum_i length(s)^{length(s)-i}},$$

otherwise.

Intuitively, `Project` tries to map the traces by splitting them according to the occurrence of the feature ϕ. Indeed, the lowest value (cf. 0) is assigned in the case where a occurs in the trace s, while the highest value (cf. 1) is assigned if all the nodes in $\{\mathtt{a}_1, \ldots, \mathtt{a}_h\}$ are in s, but a is not. Otherwise, i.e., if $\mathtt{a} \notin tasks(s)$ but some node in $\{\mathtt{a}_1, \ldots, \mathtt{a}_h\}$ is not in s, the value is obtained by lexicographically weighting the occurrences of such nodes, by giving preference to those occurring first in the control flow.

Example 5.5. Recall that $[\mathtt{abefi}] - \not\rightarrow_{\langle\sigma,0\rangle} \mathtt{k}$ and $\phi_2 : [\mathtt{acdgi}] - \not\rightarrow_{\langle\sigma,0\rangle} \mathtt{j}$, for $\sigma < 5/16$, are discriminant rules in the example in Fig. 1. Then, it is not difficult to check that $\phi_1 : [\mathtt{efi}] - \not\rightarrow_{\langle\sigma,0\rangle} \mathtt{k}$ and $\phi_2 : [\mathtt{dgi}] - \not\rightarrow_{\langle\sigma,0\rangle} \mathtt{j}$ are minimal discriminant rules, which allow to project the log into a two-dimensional vectorial space, as shown in Fig. 6a.

By looking at this feature space in Fig. 6b, four clusters can be identified: $C_1 = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$, $C_2 = \{s_8, s_9, s_{10}\}$, $C_3 = \{s_{11}, s_{12}\}$, and

$$C_4 = \{s_{13}, s_{14}, s_{15}, s_{16}\}.$$

Let us now focus on the algorithm `FindFeatures` for computing the set of all the minimal features. The algorithm takes in input a log $\mathcal{L}_P$, a workflow schema $\mathcal{WS}$, the thresholds σ and γ in Definition 5.3, a number ℓ bounding the length of the features to be discovered, and a number $maxF$ bounding the number of features that should be returned as output. The algorithm exploits a level-wise search in the space of all the possible features, in the a priori style (see, e.g., [32], [33], [34]).

To see how this exploration is possible, consider two strings $s : \mathtt{a}_1 \ldots \mathtt{a}_h$ and $s' : \mathtt{a}'_1 \ldots \mathtt{a}'_{h'}$. We say that *s directly precedes* s', denoted by $s \prec s'$, if s is a prefix of s' and $|tasks(s)| = |tasks(s')| - 1$. Moreover, we say that *s precedes* s', denoted by $s \prec^* s'$, if either $s \prec s'$ or there exists a string s'' such that $s \prec^* s''$ and $s'' \prec^* s$. It is not difficult to see that all frequent sequences can be constructed by means of a chain over the $\prec$ relation and, therefore, the space of frequent sequences forms a lower semilattice that can be explored in a bottom-up fashion: At each iteration of the main loop (Steps 3-15), the algorithm generates all the possible σ-frequent sequences whose length is len by exploiting previously computed σ-frequent sequences with length $len - 1$ and stores them in $Cand_{len}$. In particular, in Steps 8 and 9, it scans the log for singling out the sequences in $Cand_{len}$ that are σ-frequent and γ-frequent in $\mathcal{L}_P$ by storing them in the sets L^{σ}_{len} and L^{γ}_{len}, respectively. Actually, the computation starts in Step 1, where L^{σ}_2 is initialized to contain all the σ-frequent sequences of length 2. Then, candidates of length len are obtained by combining any sequence of the form $\mathtt{a}_1 \ldots \mathtt{a}_j$ in L^{σ}_{len-1} with any sequence of the form $\mathtt{a}_j\mathtt{a}$ in L^{σ}_2.

Finally, after the frequent sequences are discovered, in Steps 10-13, the features consisting of len nodes are identified and stored in F_{len}. To this purpose, the sequence of the form $\mathtt{a}_1 \ldots \mathtt{a}_j\mathtt{a}$ must be not γ-frequent and the minimality conditions must be satisfied (Step 11). At the end of each iteration, the discovered minimal features are eventually inserted into the set $\mathcal{F}$ (Step 14). The process is repeated until no other frequent sequence is found or all the features up to the length ℓ are found.

It is worth noting that the algorithm does not directly output the set $\mathcal{F}$; rather, it invokes the function *mostRelevantFeatures*, whose aim is to select (if possible) the $maxF$ most representative in $\mathcal{F}$. This latter task is carried out for reducing the dimensionality of the vectorial space. Usually, feature selection is a very complex activity and some general-purpose techniques have been proposed in the literature (see, e.g., [31], [35]). In this case, things are simpler since the semantics of discriminant rules induces quite a natural ordering among elements in $\mathcal{F}$, consisting of preferred features having the lowest value for the threshold γ. These are the most unexpected rules. Therefore, *mostRelevantFeatures* simply returns the top-$maxF$ elements in $\mathcal{F}$ with regard to this ordering.

After the algorithm has been described, we can now state its main properties.

Theorem 5.6. *The algorithm* `FindFeatures` *is such that: 1) The main loop is repeated $\ell - 2$ times at most and $\ell - 2$ is in fact the maximum number of log scans. 2) At each iteration, L^{σ}_{len} contains the set of all the σ-frequent sequences of length len. 3) At the end of the computation, the set $\mathcal{F}$ contains the set of all the minimal features of length bounded by ℓ.*

Proof.

1. By conditions in Step 3, there are at most $\ell - 2$ iterations. Moreover, for each iteration, only

```
Input: A log $\mathcal{L}_P$, a schema $\mathcal{WS} = \langle A, E, a_0, A_F, \mathit{Fork}, \mathit{Join} \rangle$, thresholds $\sigma$ and $\gamma$, natural number $\ell$ and $maxF$.
Output: A set of minimal discriminant rules.
Method: Perform the following steps:
1  $L_2^\sigma := \{\mathtt{ab} \mid \mathtt{ab}$ is $\sigma$-frequent in $\mathcal{L}_P\}$;
2  $len := 3$;   $\mathcal{F} := \emptyset$;
3  while $len \leq \ell$ and $L_{len}^\sigma \neq \emptyset$ do  //iterate on feature length
4    $Cand_{len} := \emptyset$; $F_{len} := \emptyset$;
5    for each sequence $\mathtt{a_1...a_j} \in L_{len-1}^\sigma$ do  //build candidates
6      for each $\mathtt{a_ja} \in L_2^\sigma$ do
7        $Cand_{len} := Cand_{len} \cup \{\mathtt{a_i...a_ja}\}$;
8    $L_{len}^\sigma := \{s \mid s \in Cand_{len} \wedge s$ is $\sigma$-frequent in $\mathcal{L}_P\}$;
9    $L_{len}^\gamma := \{s \mid s \in Cand_{len} \wedge s$ is $\gamma$-frequent in $\mathcal{L}_P\}$;
10   for each $\mathtt{a_1...a_ja} \in (Cand_{len} - L_{len}^\gamma)$ do  //update features
11     if $\nexists \mathtt{a_1...a_jb} \in (Cand_{len} - L_{len}^\gamma)$ s.t. $\mathtt{ab} \in L_2^\sigma$ and
          $\nexists [\mathtt{c_1...c_k}] \not\rightarrow_{\langle\sigma,\gamma\rangle} \mathtt{a}$ in $\mathcal{F}$, s.t. $\{\mathtt{c_1}, ..., \mathtt{c_k}\} \subseteq \{\mathtt{a_1}, ..., \mathtt{a_j}\}$
12     then  $F_{len} := F_{len} \cup \{[\mathtt{a_1...a_j}] \not\rightarrow_{\langle\sigma,\gamma\rangle} \mathtt{a}\}$;
13   end for
14   $\mathcal{F} := \mathcal{F} \cup F_{len}$;   $len := len + 1$;
15 end while
16 return $mostRelevantFeatures(\mathcal{F}, maxF)$;
```

Fig. 7. **Algorithm** `FindFeatures`.

Steps 8 and 9 require access to the log $\mathcal{L}_P$, which can be done by one scan only.

2. L_{len}^σ trivially contains σ-frequent sequences only, because of Step 8. Moreover, all the sequences in L_{len}^σ have length *len*; indeed, sequences in L_2^σ have, by construction, length 2 and each sequence in L_{len}^σ is the result of the merging of a sequence $\mathtt{a_1 \ldots a_j}$ (of length $len - 1$) and $\mathtt{a_ja}$—formally, one can prove this property by induction. Thus, we have to show that L_{len}^σ, in fact, contains all the σ-frequent sequences of length *len*. The proof is by structural induction on the length *len*. **Base:** For $len = 2$, the property holds by construction. **Induction:** Assume that L_{len-1}^σ contains all the σ-frequent sequences of length $len - 1$. Clearly, sequences in L_{len}^σ can be written by adding to any sequence in L_{len-1}^σ exactly one activity because of the fact that the space of such sequences forms a lower semi-lattice with regard to the "$\prec$" relation. This is precisely what is done in Step 7.
3. We show that F_{len} contains exactly the minimal features of size *len*, for $len \geq 3$. By construction in Steps 10-12, F_{len} contains only minimal features of size *len*. Therefore, we have to show that every feature ϕ of size *len* of the form $[\mathtt{a_1 \ldots a_j}] - \not\rightarrow_{\langle\sigma,\gamma\rangle} \mathtt{a}$ is in F_{len}. To this aim, it is sufficient to exploit the completeness result in point (2) above, and notice that $\mathtt{a_1 \ldots a_j}$ is in L_{len-1}^σ and $\mathtt{a_ja}$ is in L_2^σ. □

Now that all the procedures in the `ProcessDiscovery` algorithm have been discussed, we can explicitly note that it requires a number of scans on the log which linearly depends on the parameters ℓ, k, and m. This is a nice property for scaling in real applications. To see why this is the case, recall that, by Theorem 5.1, the algorithm makes m iterations at most. At each iteration, the dominant operation is the procedure *refineWorkflow* in which there is one invocation of `FindFeatures` (for computing features of length ℓ) for each of the k different invocations of the algorithm `MineWorkflow`. Then, by combining results in Theorem 5.6 and Theorem 4.3, the following is obtained.

Corollary 5.7. *Algorithm* `ProcessDiscovery` *requires* $O(m \times k \times \ell)$ *scans of the log* L_P.

6 EXPERIMENTAL RESULTS

All the algorithms proposed in the paper have been implemented and recently integrated in the *ProM* process mining framework [36] as an *analysis* plug-in,[2] where the user is allowed to exploit any available mining algorithm to equip each cluster with a schema.

In this section, we illustrate the results of experimental activity aimed at assessing the practical effectiveness of the proposed approach, the theoretical guarantee on the scaling (cf. Corollary 5.7), and the effectiveness in deriving conformant models (cf. Theorem 5.1).

6.1 Qualitative Results

We start our analysis by discussing the results of `ProcessDiscovery` on some example scenarios. We firstly considered the *OrderManagement* process, and we randomly generated $5,000$ traces for the workflow schema in Fig. 1. Notably, in the generation of the log, we also required that task `k` could not occur in any trace containing

2. Available at http://www.icar.cnr.it/wfmining.

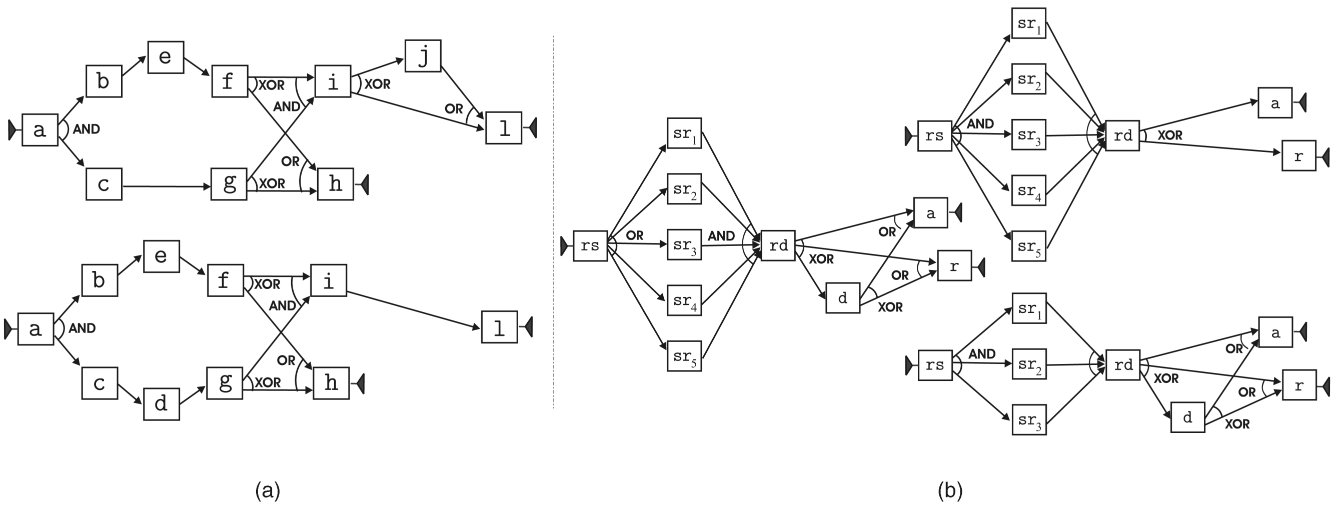


Fig. 8. Results for the `ProcessDiscovery` algorithm on example processes.

e, and that task j could not appear in any trace containing d, thereby modeling the restriction that a fidelity discount is never applied to a new customer and that a fast dispatching procedure cannot be performed whenever some external supplies were asked for.

As a matter of fact, these constraints determine different usage scenarios, which are mixed up in the schema in Fig. 1, thereby leading to a low value of soundness. For instance, this schema admits that a discount is applied to a new customer.

Conversely, by running `ProcessDiscovery` with $maxFeature = 5$, $maxLevels = 1$, $k = 4$, $\sigma = 0.05$, $\gamma = 0.01$, and $\ell = 5$, the four schemas associated with the discovered clusters are the two of Fig. 2 plus the two reported in Fig. 8a. Each of these schemas captures, in fact, exactly one of the possible usage scenarios for the process (which are determined by the type of customer and the availability in stock) and models one of the clusters discovered in Example 5.5. The resulting mined model is both 1-sound and 1-complete and yields a clearer picture of the behavior and of the organizational rules of the process.

As a further example, let us consider the *ReviewPaper* process of handling the revisions for a paper submitted to a scientific conference. The process consists of the following tasks: (rs) receiving the submission, ($\mathtt{sr}_i$, $1 \leq i \leq 5$), sending the paper to the reviewers, (rd) receiving the revisions and taking a decision, (d) discussing on the paper in the case revisions are not uniform, (a) accepting the paper, and (r) rejecting the paper. Actually, in the case where the paper is authored by a program committee member, it has to be reviewed by five reviewers and it is immediately rejected in the case in which a reviewer does not want it to be accepted for publication. Otherwise, only three reviewers are assigned to the paper. A possible workflow schema for the process is reported on the left of Fig. 8b—notice, e.g., that rs is an *or-fork*. According to it and to the above specified rules, a log (of 5,000 traces) was randomly generated and `ProcessDiscovery` was invoked with $maxFeature = 2$, $maxLevels = 1$, $k = 2$, $\sigma = 0.05$, $\gamma = 0.01$, and $\ell = 5$. The two resulting schemas are shown on the right of the same figure. It should be clear that one schema is, in fact, the 1-sound model for handling the revision of a paper written by a program committee member, while the other is the 1-sound model for handling the revision of all the other papers—notice, e.g., that, for both schemas, rs is an *and-fork* task, now.

The examples above evidenced that the clustering technique is very effective in providing insights into a process, especially in the case where the enactments are constrained by some kinds of behavioral rule, possibly involving information which is beyond the pure execution of activities (e.g., stored in some database). This is quite a common situation in practical applications. In fact, research in modeling languages already evidenced the importance of these properties that cannot be captured by a graph model and that, in the current workflow management systems, are either left unexpressed or modeled by means of some form of logics.

We conclude by noticing that, even in the case where no behavioral rules are defined and, hence, there is only one usage scenario, the `ProcessDiscovery` algorithm is still useful in order to identify some (hidden) variants which correspond to anomalies and malfunctioning in the system. In these cases, the effect of the algorithm is to identify the "normal" behavior of the process and to single out the instances that are deviant with regard to it.

This intuition has been confirmed by several experiments on real data sets available at http://www.processmining.org. For the sake of completeness, in Fig. 9, we report the hierarchy built for an example log file (cf. *a12f5n20.xml*, $maxFeature = 3$, $k = 3$, $\sigma = 0.01$, $\gamma = 0.4$, and $\ell = 5$) and the models associated with each node in the first level of the hierarchy. The algorithm discovered one large cluster $R0$ whose schema coincides with the one the root R is equipped with. Moreover, clusters $R1$ and $R2$ (containing 17 and 37 traces, respectively) may, indeed, be perceived as outliers with regard to the discovered main behavior.

6.2 Quantitative Results

Besides qualitative tests, we performed an extensive experimentation on synthetic logs, produced by means of a generator which takes advantage of ideas exposed in [33] and extends the one described in [23]. Notably, the generator can be used to produce a log of traces which are compliant with a given workflow schema, as well as to generate random workflow schemas.

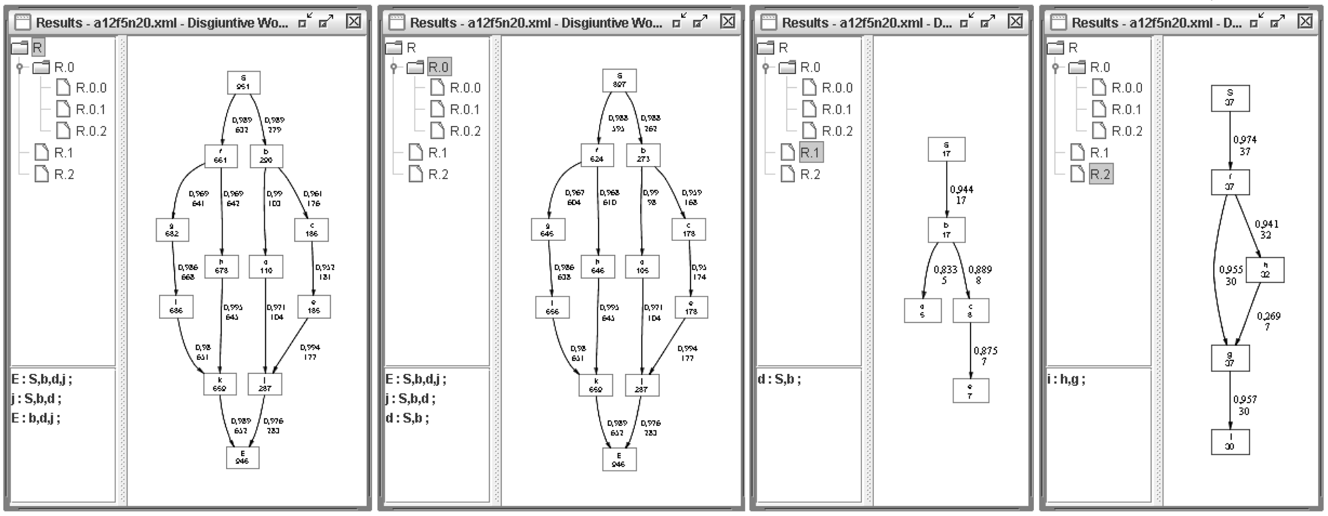


Fig. 9. Results on log data available at http://www.processmining.org.

6.2.1 Text Procedure

In order to assess the effectiveness of the technique, we defined a simple test procedure for comparing available workflow models with the output of the `ProcessDiscover` algorithm. Let $\mathcal{WS}$ be a given workflow schema and $\mathcal{L}_P$ be a log of traces compliant with $\mathcal{WS}$, produced by means of the generator. Then, the result of each test is a disjunctive workflow schema $\mathcal{WS}^{\vee}$ extracted by providing the `ProcessDisovery` algorithm with $\mathcal{L}_P$ as input. Notice that, for evaluating the quality of a mined schema, we are only interested in computing its soundness, since it was proven, by Theorem 5.1, that any schema discovered by means of our technique has maximal completeness. Specifically, the soundness of $\mathcal{WS}^{\vee}$ is estimated by computing the percentage of the traces in a log L_{test} (randomly generated from $\mathcal{WS}^{\vee}$) that are also compliant with the original schema $\mathcal{WS}$. Ideally, when L_{test} contains all the possible traces of $\mathcal{WS}^{\vee}$, the estimate and the actual value coincide.

In light of Corollary 5.7, the experiments mainly focus on the influence of the branching factor k and on the number of levels ($maxLevels$) in the hierarchy of clusters—notice that $maxLevels$ and k in fact determine the size of the mined disjunctive schema (m). The parameter ℓ, i.e., the length of the features, is instead kept fixed to 5 since, after several experiments, this appeared to be a good compromise between running time and quality of results. And, in fact, for $\ell > 5$, the soundness of the mined schema rarely improves.

All the tests were conduced on a 1600MHz/256MB Pentium IV machine running Windows XP Professional.

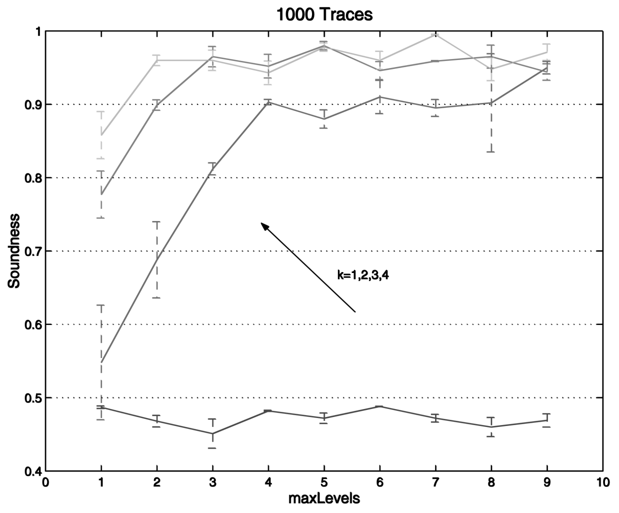
6.2.2 Results

A first set of experiments was conducted to assess the soundness of the mined models. To this purpose, we considered a fixed workflow schema at time and some log traces randomly generated according to it. This first set of experiments was repeated for different synthetic and real schemas and the algorithm performed quite similarly under all the circumstances. Moreover, for each schema, we generated a number of different training logs. Here, we report the average values and their associated standard deviation.

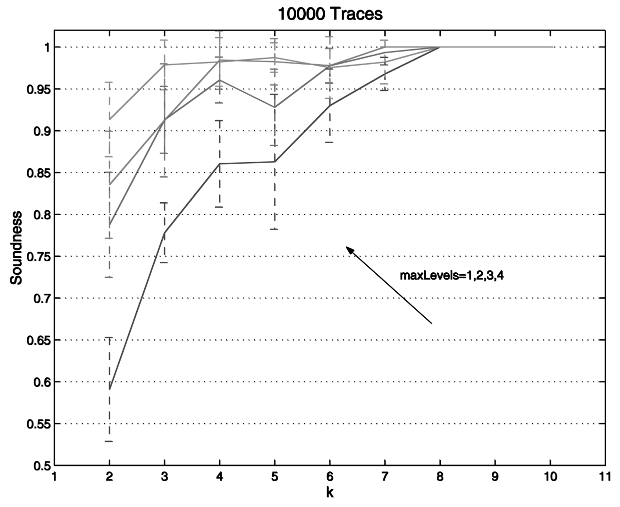
For a process with 40 tasks, the soundness values of the mined model are graphically illustrated in Fig. 10, which reports the mean and standard deviation of the soundness obtained with different values of k and $maxLevels$ over input logs consisting of 1,000 traces (on the left) or 10,000 traces (on the right). Both charts in the figure show that the quality of the mined schema generally gets better when either of these parameters increases, up to achieving the maximum value of soundness. It is interesting to express a further remark about the way the branching factor k impacts on soundness. As a matter of fact, observe that, for $k = 1$, the algorithm degenerates in computing a unique schema and, hence, the soundness is not affected by the parameter $maxLevels$. On the contrary, for $k > 1$, the algorithm is able to rediscover the original schema after performing a suitable number of iterations—see, in particular, Fig. 10b.

Notice that the case where $k = 1$ is that of all classical process mining algorithms (see the next section) because no clustering is performed. However, since the aim of the paper is to demonstrate the effectiveness of the clustering approach (independently of the way `MineWorkflow` is carried out), we are not interested in a thorough comparison here. We only evidence that, while we often get a low value of soundness in the case where $k = 1$, mined schemas are always guaranteed to be 1-complete no matter what the log, whereas most of the previous algorithms either assume that the log itself is complete or accept incompleteness as a manifestation of noise. Thus, these approaches in process mining are orthogonal and enjoy specific advantages and their comparison is left as a subject for further research.

A second set of experiments was aimed at providing more insight on the impact of k on effectiveness and efficiency. Input data are the same as those used for above experiments. Fig. 11 shows the results. Specifically, the left side of the figure confirms that k strongly impacts the soundness, almost independently of the log size $|\mathcal{L}_P|$, provided that it is big enough to reduce the effects of statistical fluctuations in the log composition. Moreover, Fig. 11b shows that the total time needed for building a schema also increases when we use higher values for either k or $maxLevels$. However, as expected, the scaling is linear in both parameters. In particular, higher values for

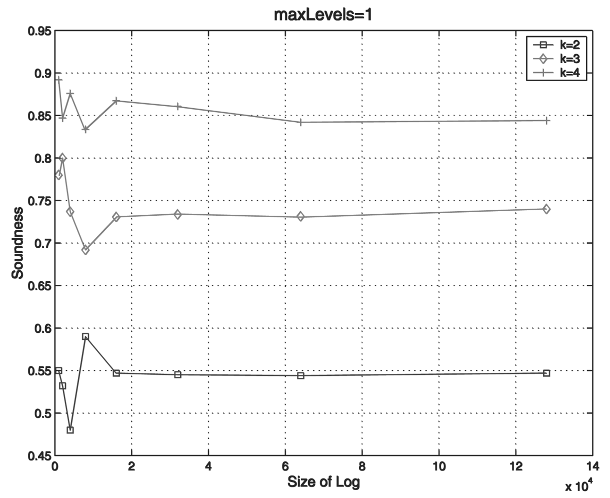


(a)

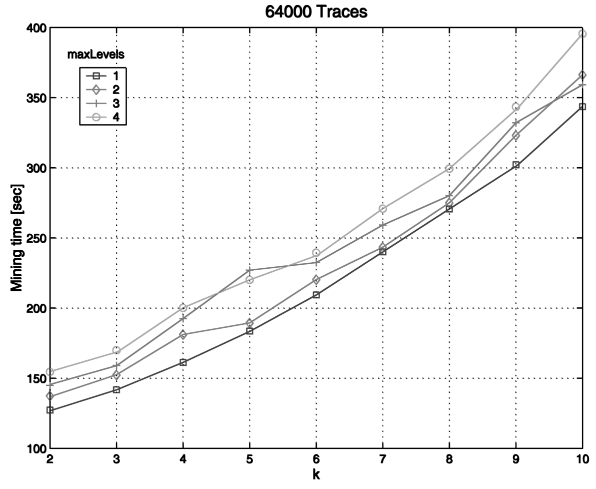


(b)

Fig. 10. Fixed workflow: Soundness with regard to (a) number of levels and (b) k.



(a)



(b)

Fig. 11. Fixed workflow: (a) Soundness and (b) mining time with regard to k.

$maxLevels$ only mildly increase the running time; this observation and the behavior shown in Fig. 10a are arguments in favor of using small values for k (see the discussion in Section 5.1).

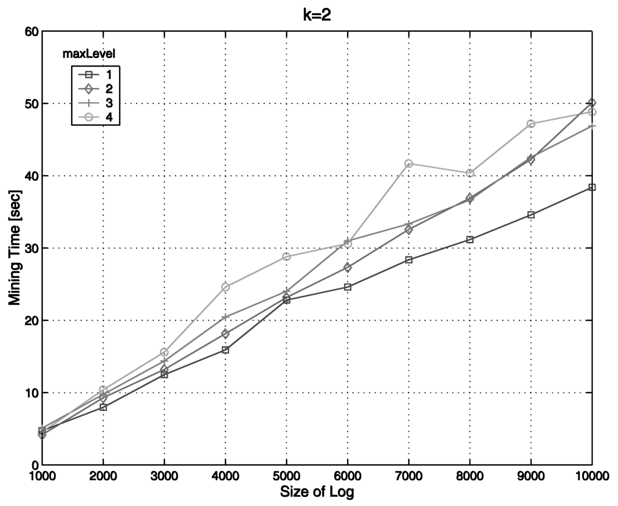
In Fig. 12, we report the trend of the mining time with respect to the log size. In particular, in Fig. 12a, several curves are plotted which correspond to different values of $maxLevels$, with a fixed branching factor ($k = 2$); analogously, in Fig. 12b, it is k to be varied, while $maxLevels$ is always set to 2. All these curves substantiate the good scalability of the approach, which takes a time that depends linearly on the number of traces used as input.

In the last series of experiments, we randomly generated several workflow schemas with a different number of tasks and precedence relationships. In Fig. 13, we report the results obtained with four workflow schemas. Observe in Fig. 13a that, for a fixed value of k, the soundness of the mined schema tends to become lower as the schema complexity augments, i.e., the number of activities, links, and constraints increases. This testifies to the fact that, in order to have 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 Fig. 13b.

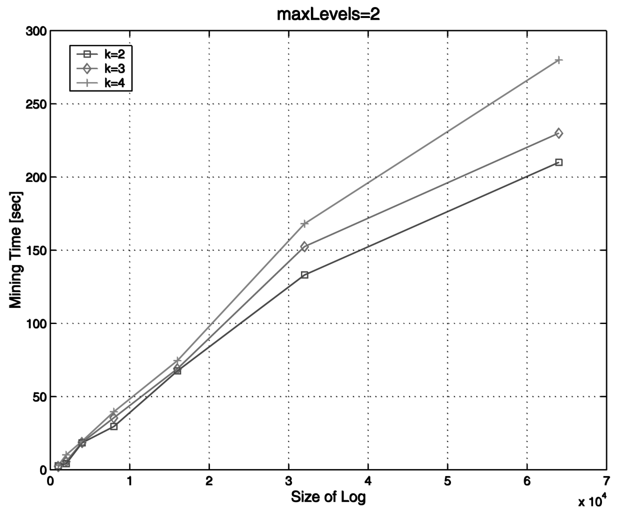
7 Overview of Process Mining Algorithms

We next briefly review some previous works on process mining, which constituted a fundamental source of inspiration for our research. In fact, these approaches solve the $\mathrm{MPD}(\mathcal{L}_P, 1)$ problem and may be even used in our algorithm in place of `MineWorkflow`. A broader, and up-to-date enough, overview on this topic can be found in [1].

Process mining was first introduced in a software engineering setting by [37] and subsequently extended in [5], [38]. The paper proposes three techniques, namely, 1) a statistical approach, based on neural networks, 2) a purely algorithmic approach, and 3) a hybrid approach, based on Markov models, for automatically deriving a formal model from execution's log. The model is a Finite State Machine

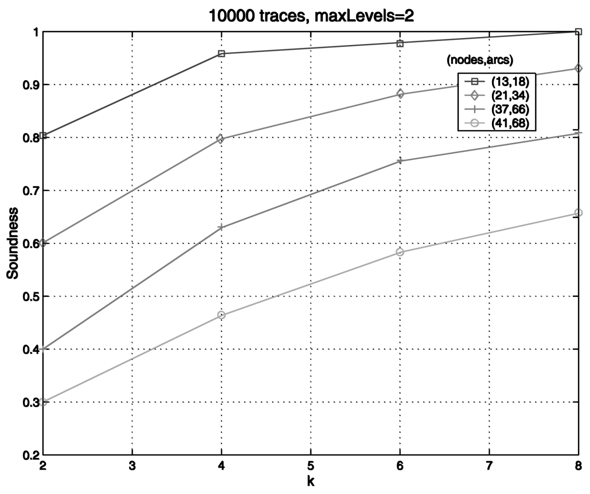


(a)

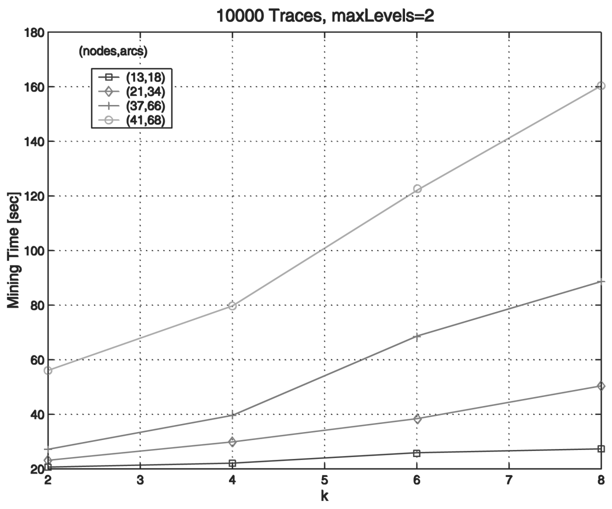


(b)

Fig. 12. Fixed workflow: Scalability with regard to log size, varying the (a) number of levels and (b) k.



(a)



(b)

Fig. 13. Variable workflow: (a) Soundness and (b) mining time with regard to k.

(FSM) model, where the activities are associated with the edges and specify transitions between states.

In [8], processes are more naturally represented through pure directed graphs, which are allowed to express precedence relationships only, by disregarding richer control flow constructs, such as concurrency, synchronization, and choice. Notably, techniques in [8] have been designed to work even in the presence of cyclic dependencies between activities.

A special kind of Petri nets, named Workflow nets (WF-net), was adopted in [39], [40], [1], [21] for modeling and mining workflow processes. There, each transition represents a task, while the relationships between the tasks are modeled by arcs and places. Importantly, WF-nets allows for recursive schemas and iterated executions. A basic algorithm, called α-algorithm, is introduced in [21], which is able to derive a WF-net from a workflow log, under the assumption that the log is complete and free of noise. However, the α-algorithm can easily deal with cycles, which is a functionality currently missing in our approach. Actually, the capability of the algorithm to mine WF-net workflow models and its limitations are analyzed in [1], where the concept of *structured workflow (SWF) net* is introduced to capture a class of WF-nets which a process mining algorithm should be able to rediscover. Some extensions to the approach are presented which address these problems. Specifically, in [40], simple statistics are exploited in the construction of the ordering relations in order to cope with noise in the logs, whereas specific preprocessing and postprocessing strategies for capturing short loops are devised in [41].

A further approach to mining a process model from event logs is described in [7], [42], where a subset of the ADONIS language [43] is adopted to represent a process model. An important peculiarity of the approach mainly resides in its capability of recognizing duplicate tasks in the control flow graph, i.e., many nodes associated with the same task.

Yet another approach is adopted in [9], where a mining tool is presented which is able to discover hierarchically structured workflow processes. Such a model corresponds to an expression tree, where the leaves represent tasks (operands) while any other node is associated with a control flow operator. In this context, the mining algorithm mainly consists of a suitable set of term rewriting systems.

The possibility of grouping workflow traces has been previously explored by [11]. The basic idea is to exploit a multiphase process mining approach, where individual models are first generated for each process instance by means of the technique in [10] and, subsequently, aggregated into *aggregation graphs*. These graphs may be eventually translated into EPCs or Petri nets. Interestingly, the first phase can be possibly avoided in the case where logs are registered by some tools such as ARIS PPM [44], where individual models are already available in terms of *instance graphs* and do not need to be preliminary mined.

As a matter of fact, the technique in [11] can be used to aggregate any predetermined set of instance graphs and, therefore, it does not fit our setting where the traces to be aggregated are not known in advance (in fact, clusters are the result of an automatic partitioning on the log based on some suitably extracted features). Yet, the grouping technique can still be used to implement some kinds of *agglomerative* clustering algorithm. Indeed, an algorithm may start with each instance graph associated with an aggregation graph and, at each step, it may select the most similar aggregation graphs (according to some suitably defined metric) whose instances are in turn fused into a more general aggregation graph. In this way, the hierarchy of schemas may be defined in a bottom-up fashion, rather than in a top-down one. We believe that the above sketched approach is worth formalizing and analyzing in order to assess its practical effectiveness.

8 Conclusions

In this paper, we have proceeded to investigate data mining techniques to discover process models from event logs. We have devised a novel framework that substantially differs from previous approaches for it performs a hierarchical clustering of the logs in which each trace is seen as a point of a properly identified space of features. The resulting model is a disjunctive schema that explicitly takes care of variants of the process. The computational complexity of the different problems involved in our investigation has been thoroughly investigated. It turned out that even checking whether a given model at hand is conformant is a difficult task so that any efficient algorithm for computing the best schema has to search into the space of the possible solutions by never checking for the soundness. Our solution to this problem was a smart algorithm that stepwise refines a starting schema by guaranteeing that each refinement leads to an increasingly sound schema. The performances of the proposed approach were analyzed over a number of data sets, thereby getting some appreciable evidence for its effectiveness and scalability.

Notably, the proposed approach is, to a large extent, independent of the adopted workflow model and is, indeed, modular and able to benefit from other results available in the literature for dealing with more elaborate features, such as cyclicity in the log. And, in fact, our clustering algorithm has been made recently available as an *analysis* plug-in for the *ProM* process mining framework, which allows for the exploitation of well-established process mining algorithms to equip clusters with models. An extensive investigation of this kind of integrated approach constitutes an avenue of further research.

We conclude by observing that the whole framework proposed in the paper is essentially *propositional* as it assumes a simplification of the schema and of the enactments in which many real-life details are omitted. This is a standard assumption in current research in process mining. Therefore, another interesting avenue for further research is to extend our techniques to take care of the environment so that clusters may reflect not only structural similarities among traces, but also information about, e.g., users and data values.

References

[1] 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 Eng.*, vol. 47, no. 2, pp. 237-267, 2003.

[2] A. Rozinat and W.M.P. van der Aalst, "Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models," *Proc. Int'l Workshop Business Process Intelligence (BPI '05)*, pp. 1-12, 2005.

[3] F. Casati, M. Castellanos, and M. Shan, "Enterprise Cockpit for Business Operation Management," *Proc. 23rd Int'l Conf. Conceptual Modeling (ER '04)*, pp. 825-827, 2004.

[4] D.-R. Liu and M. Shen, "Workflow Modeling for Virtual Processes: An Order-Preserving Process-View Approach," *Information Systems*, vol. 28, pp. 505-532, 2003.

[5] J.E. Cook and A.L. Wolf, "Event-Based Detection of Concurrency," *Proc. Sixth Int'l Symp. Foundations of Software Eng. (FSE '98)*, pp. 35-45, 1998.

[6] A.J.M.M. Weijters and W.M.P. van der Aalst, "Rediscovering Workflow Models from Event-Based Data Using Little Thumb," *Integrated Computer-Aided Eng.*, vol. 10, no. 2, pp. 151-162, 2003.

[7] J. Herbst, "Dealing with Concurrency in Workflow Induction," *Proc. European Concurrent Eng. Conf.*, 2000.

[8] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining Process Models from Workflow Logs," *Proc. Sixth Int'l Conf. Extending Database Technology (EDBT '98)*, pp. 469-483, 1998.

[9] G. Schimm, "Mining Most Specific Workflow Models from Event-Based Data," *Proc. Int'l Conf. Business Process Management*, pp. 25-40, 2003.

[10] B.F. van Dongen and W.M.P. van der Aalst, "Multi-Phase Process Mining: Building Instance Graphs," *Proc. Int'l Conf. Conceptual Modeling (ER)*, pp. 362-376, 2004.

[11] B.F. van Dongen and W.M.P. van der Aalst, "Multi-Phase Process Mining: Aggregating Instance Graphs into Epcs and Petri Nets," *Proc. Int'l Workshop Applications of Petri Nets to Coordination, Worklflow and Business Process Management (PNCWB) at ICATPN '05*, 2005.

[12] H. Davulcu, M. Kifer, C.R. Ramakrishnan, and I.V. Ramakrishnan, "Logic Based Modeling and Analysis of Workflows," *Proc. 17th ACM Symp. Principles of Database Systems (PODS '98)*, pp. 25-33, 1998.

[13] P. Muth, J. Weifenfels, M. Gillmann, and G. Weikum, "Integrating Light-Weight Workflow Management Systems within Existing Business Environments," *Proc. 15th IEEE Int'l Conf. Data Eng. (ICDE '99)*, pp. 286-293, 1999.

[14] P. Senkul, M. Kifer, and I.H. Toroslu, "A Logical Framework for Scheduling Workflows under Resource Allocation Constraints," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02)*, pp. 694-702, 2002.

[15] H. Schuldt, G. Alonso, C. Beeri, and H. Schek, "Atomicity and Isolation for Transactional Processes," *ACM Trans. Database Systems*, vol. 27, no. 1, pp. 63-116, 2002.

[16] W.M.P. van der Aalst, A. Hirnschall, and H.M.W. Verbeek, "An Alternative Way to Analyze Workflow Graphs," *Proc. 14th Int'l Conf. Advanced Information Systems Eng.*, pp. 534-552, 2002.

[17] M. Kamath and K. Ramamritham, "Correctness Issues in Workflow Management," *Distributed Systems Eng.*, vol. 3, no. 4, pp. 213-221, 1996.

[18] W.M.P. van der Aalst, "The Application of Petri Nets to Worflow Management," *J. Circuits, Systems, and Computers,* vol. 8, no. 1, pp. 21-66, 1998.

[19] W.M.P. van der Aalst, J. Desel, and E. Kindler, "On the Semantics of EPCS: A Vicious Circle," *Proc. EPK 2002: Business Process Management Using EPCs,* pp. 71-80, 2002.

[20] G. Keller, M. Nuttgens, and A.W. Scheer, *Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK).* Univ. of Saarland, Saarbrucken, 1992.

[21] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 9, pp. 1128-1142, Sept. 2004.

[22] L. Pontieri, G. Greco, A. Guzzo, and D. Sacca, "Discovering Expressive Process Models by Clustering Log Traces [Appendix]," http://www.icar.cnr.it/wfmining, 2005.

[23] G. Greco, A. Guzzo, G. Manco, and D. Sacca, "Mining and Reasoning on Workflows," *IEEE Trans. Data and Knowledge Eng.,* vol. 17, no. 4, pp. 519-534, Apr. 2005.

[24] G. Greco, A. Guzzo, and L. Pontieri, "Mining Hierarchies of Models: From Abstract Views to Concrete Specifications," *Proc. Int'l Conf. Business Process Management,* pp. 32-47, 2005.

[25] R.J. van Gabbeek and W.P. Weijland, "Branching Time and Abstraction in Bisimulation Semantics," *J. ACM,* vol. 43, no. 3, pp. 555-600, 1996.

[26] T. Basten and W. van der Aalst, "Inheritance of Workflows: An Approach to Tackling Problems Related to Change," *Theoretical Computer Science,* vol. 270, nos. 1-2, pp. 125-203, 2002.

[27] V. Guralnik and G. Karypis, "A Scalable Algorithm for Clustering Sequential Data," *Proc. IEEE Int'l Conf. Data Maning (ICDM '01),* pp. 179-186, 2001.

[28] J. Han, J. Pei, B. Mortazavi-Asl, U. Dayal, and M. Hsu, "Freespan: Frequent Pattern-Projected Sequential Pattern Mining," *Proc. Int'l ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '00),* pp. 355-359, 2000.

[29] Y.S. Kim, W.N. Street, and F. Menczer, "Feature Selection in Unsupervised Learning via Evolutionary Search," *Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '00),* pp. 365-369, 2000.

[30] N. Lesh, M.J. Zaki, and M. Ogihara, "Mining Features for Sequence Classification," *Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '00),* pp. 342-346, 1999.

[31] H. Motoda and H. Liu, "Data Reduction: Feature Selection," *Handbook of Data Mining and Knowledge Discovery,* pp. 208-213, 2002.

[32] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *Proc. ACM SIGMOD,* pp. 207-216, 1993.

[33] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Databases,* pp. 487-499, 1994.

[34] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. 11th Int'l Conf. Data Eng. (ICDE '95),* pp. 3-14, 1995.

[35] B. Padmanabhan and A. Tuzhilin, "Small Is Beautiful: Discovering the Minimal Set of Unexpected Patterns," *Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '00),* pp. 54-63, 2000.

[36] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst, "The Prom Framework: A New Era in Process Mining Tool Support," *Proc. 26th Int'l Conf. Applications and Theory of Petri Nets (ICATPN '05),* pp. 444-454, 2005.

[37] J.E. Cook and A.L. Wolf, "Automating Process Discovery through Event-Data Analysis," *Proc. 17th Int'l Conf. Software Eng. (ICSE '95),* pp. 73-82, 1995.

[38] J.E. Cook and A.L. Wolf, "Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model," *ACM Trans. Software Eng. Methodology,* vol. 8, no. 2, pp. 147-176, 1999.

[39] W.M. P. van der Aalst and K.M. van Hee, *Workflow Management: Models, Methods, and Systems.* MIT Press, 2002.

[40] W.M. P. van der Aalst and B.F. van Dongen, "Discovering Workflow Performance Models from Timed Logs," *Proc. Int'l Conf. Eng. and Deployment of Cooperative Information Systems (EDCIS '02),* pp. 45-63, 2002.

[41] A.K.A de Medeiros, B.F. van Dongen, W.M.P. van der Aalst, and A.J.M.M. Weijters, "Process Mining: Extending the A-Algorithm to Mine Short Loops," Technical Report, Univ. of Technology, Eindhoven, BETA Working Paper Series, WP 113, 2004.

[42] J. Herbst and D. Karagiannis, "Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models," *J. Intelligent Systems in Accounting, Finance, and Management,* vol. 9, pp. 67-92, 2000.

[43] S. Junginger, H. Kuhn, R. Strobl, and D. Karagiannis, "Ein Geschafts-Prozessmanagement-Werkzeug der Nachsten Generation—adonis: Konzeption und Anwendungen," *Wirtschaftsinformatik,* vol. 42, no. 3, pp. 392-401, 2000.

[44] I.D.S. Scheer, "Aris Process Performance Manager (Aris PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper)," http://www.ids-scheer.com, 2002.

Gianluigi Greco received the Laurea degree in computer science engineering from the University of Calabria, Italy, in 2000. Currently, he is an assistant professor of computer science in the Department of Mathematics at the University of Calabria. His main research interests are across the areas of databases and artificial intelligence and focus on database theory, knowledge representation, nonmonotonic reasoning, computational complexity, and deductive databases.

Antonella Guzzo received the Laurea degree in engineering from the University of Calabria and the PhD degree in system engineering and computer science from the DEIS Department at the University of Calabria. Since January 2005, she has been a research fellow at the High Performance Computing and Networks Institute (ICAR-CNR) of the National Research Council of Italy. Her research interests include workflow management, process mining, data mining, and knowledge representation.

Luigi Pontieri received the Laurea degree in computer engineering in July 1996 and the PhD in system engineering and computer science in April 2001 from the University of Calabria. He is currently a senior researcher at the High Performance Computing and Networks Institute (ICAR-CNR) of the National Research Council of Italy and a contract professor at the University of Calabria, Italy. His current research interests include information integration, data mining, and process mining.

Domenico Saccà received the doctoral degree in engineering from the University of Rome in 1975. Since 1987, he has been a full professor of computer engineering at the University of Calabria and, since 1995, he has also been director of the CNR (the Italian National Research Council) Research Institute ICAR (Institute for High Performance Computing and Networking). In the past, he was a visiting scientist in the IBM Laboratory of San Jose, California, in the Computer Science Department at the University of California at Los Angeles, and in the ICSI Institute of Berkeley, California; furthermore, he was a scientific consultant at MCC, Austin, and manager of the Research Division of CRAI. His current research interests focus on advanced issues of databases such as scheme integration in data warehousing, compressed representation of datacubes, workflow and process mining, and logic-based database query languages. He has been a member of the program committees of several international conferences, director of international schools and seminars, and leader of many national and international research projects. He is a member of the IEEE Computer Society.