Mining Frequent Instances on Workflows

Gianluigi Greco[1], **Antonella Guzzo**[1], **Giuseppe Manco**[2] and **Domenico Saccà**[1,2]

DEIS[1], University of Calabria, Via Pietro Bucci, 87036 Rende, Italy
ICAR-CNR[2], National Research Council, Via Pietro Bucci, 87036 Rende, Italy
{ggreco,guzzo}@si.deis.unical.it, {manco,sacca}@icar.cnr.it

Abstract. A workflow is a partial or total automation of a business process, in which a collection of *activities* must be executed by humans or machines, according to certain procedural rules. In recent years, much attention has been devoted in developing tools for workflow management, that allow the users both to specify the "static" aspects, like preconditions, precedences among activities, rules for exception handling, and to control its execution, by scheduling the activities on the available resources.
This paper deals with an aspect of workflows which has not so far received much attention even though it is crucial for the forthcoming scenarios of large scale applications on the web: providing facilities for the human system administrator to monitor the actual behavior of the workflow system in order to predict the "most probable" workflow executions. In this context, we develop a data mining algorithm for identifying frequent patterns, i.e., the workflow substructures that have been scheduled more frequently by the system. The algorithm is based on an intuitive and original graph formalization of a workflow scheme and its occurrences, used both to prove some intractability results that strongly motivate the use of data mining techniques and to derive interesting structural properties for reducing the space of search for frequent patterns. Indeed the experiments we have carried out show that our algorithm outperforms the standard approaches adapted to mining frequent instances.

Keywords: workflow management systems, frequent pattern mining.

1 Introduction

Even though the modern enterprisers increasingly use the workflow technology for designing the business process, it is surprising to observe that a little effort has been paid by the research to support workflow administrator's tasks. For instance, only in a recent paper [17], it is formally addressed the problem of finding an appropriate system configuration that guarantees a domain-specific quality of service.

Other attempts have been done in the context of process discovery [1]. The idea is to use the information collected at run-time, in order to derive a structural model of the interactions. Such a model can be used in both the diagnosis phase and the (re)design phase. In this setting the problem encompasses the one of finding sequential patterns; indeed, process graphs are richer in structure as they admit partial ordering among activities and parallel executions (see, e.g, [21, 26, 10]).

In this paper, we continue on the way of providing facilities for the human system administrator, by investigating the ability of predicting the "most probable" workflow execution. Indeed, in real world-cases, the enterprise must do many choices during workflow execution; some choices may lead to a benefit, others should be avoided in the future. Data mining techniques may, obviously, help the administrator, by looking at all the previous instantiations (collected into *log* files in any commercial system), in order to extract unexpected and useful knowledge about the process, and in order to take the appropriate decisions in the executions of further coming instances.

Consider, for instance, Figure 1 reporting a simplified schema describing the "sales ordering" process (the elements used for designing the properties of the schema have been chosen according to the standards proposed by the Workflow Management Coalition [28]). The process is as follows: a customer issues a request to purchase a given product; the enterprice checks both the availability of the required stock and the reliability of the client. Moreover, if the client is reliable but the products are partially stocked, then a production will be planned. The final states can be the acceptance or the rejection of the order.

Some applications of the discovered knowledge can be found in solving problems as follows:

- *Failure/success characterization*: by analyzing the past experience, a workflow administrator may be interested in knowing which discriminant factors characterize the failure or the success in the execution of a workflow.
- *One Step Prediction*: assume that the execution is at a given point in which the administrator has to choose an activity to start, from a given set of potential activities. Then, she/he typically wants to know which is the choice performed in the past, that more frequently had led to a desired final configuration (e.g., to the acceptance of the order).

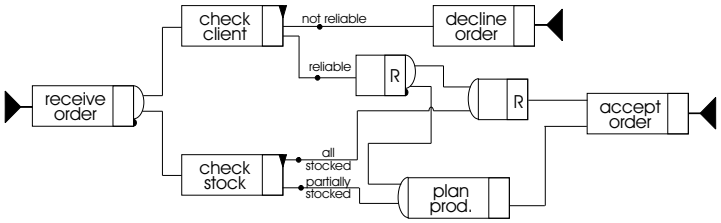

Fig. 1. Sales Ordering Process

- *General Graph Prediction*: given a set of tasks that she/he need to achieve, he wants to know the precise sequence of choices that, in past executions, had lead to the desired result.
- *Workflow Optimization*: the frequent structures characterizing the successful execution of a workflow can be profitably used to reason on the optimality (w.r.t. some real-case interesting parameter) of workflow executions.

A first step toward an automatic solution to all these problems, consists of identifying the structure of the executions, that have been scheduled more frequently by the workflow system. This problem encompasses well-known techniques, such as frequent itemsets or sequences discovery [2, 4], and can be reconduced to more involved pattern mining algorithms, that arise in complex domains.

In this context, we mention the problem of discovering frequent substructures of a given graph [22, 23], or the problem of discovering frequent subgraphs in a forest of graphs [30]. It is clear that such approaches could be well-suited to deal with the problem at hand. Indeed, we can model a workflow schema as a graph, and the executions of the workflow as a forest of subgraphs complying with the graph representing the workflow schema. However, many additional features, such as the capability of specifying constraints on the execution of a workflow, make the cited approaches unpractical both from the expressiveness and from the efficiency viewpoint. One could think also at more expressive approaches, such as the multirelational approaches, based on a first-order modeling of the features of a workflow [13]. However, in our opinion, a more effective approach can be obtained by properly formalizing the problem at hand in a suitable way.

Contribution. We investigate the problem of mining frequent workflow instances; despite its particular applicative domain, the problem we consider is of a wider interest because all the proposed techniques, can be profitably exported into other fields. The main reason of this claim is that the approach used in workflow systems is to model the process as a particular directed graph.

Throughout the paper, we propose a quite intuitive and original formalization of the main workflow concepts in terms of graph structures. This gives us the possibility to prove some intractability results in reasoning over workflows. Indeed, the in-depth theoretical analysis we provide strongly motivates the use of Data Mining techniques, thus confirming the validity of the approach. Moreover,

the formalization provides a viable mean for developing a levelwise theory which characterizes the problem of mining frequent instances of workflows.

In more detail, we propose a levelwise algorithm for mining frequent sub-graph structures (instances) that conform to a given schema (workflow specification). Several experiments confirm that our approach outperforms pre-existing techniques rearranged to this particular problem.

Organization. The paper is organized as follows. Section 2 provides a formal model of workflows, and many complexity considerations on such a proposed model. A characterization of the problem of mining frequent patterns from workflow schemas is devised in sect. 3, and in sect. 4 a levelwise theory of workflow patterns (and a corresponding algorithm for mining such patterns) is developed. Finally, sect. 5 provides an experimental validation of the approach.

2 The workflow abstract model

A *workflow schema* $\mathcal{WS}$ is a tuple $\langle A, E, A^{\odot}, A^{\vee}, E^{!}, E^{?}, E^{\subseteq}, a_0, F\rangle$, where A is a finite set of *activities*, partitioned into two disjoint subsets $A^{\odot}$ and $A^{\vee}$; $E \subseteq (A - F) \times (A - \{a_0\})$ is an acyclic relation of precedences among activities, partitioned into three pairwise disjoint subsets $E^{!}$, $E^{?}$, and $E^{\subseteq}$; $a_0 \in A$ is the starting activity, and $F \subseteq A$ is the set of final activities.

For the sake of presentation, whenever it will be clear from the context, a workflow schema $\mathcal{WS} = \langle A, E, A^{\odot}, A^{\vee}, E^{!}, E^{?}, E^{\subseteq}, a_0, F\rangle$ will also be denoted by $\langle A, E, a_0, F\rangle$ or even simpler by $\langle A, E\rangle$.

Informally, an activity in $A^{\odot}$ acts as synchronizer (also called a *join* activity in the literature), for it can be executed only after all its predecessors are completed, whereas an activity in $A^{\vee}$ can start as soon as at least one of its predecessors has been completed. Moreover, once finished, an activity a activates all its outgoing $E^{!}$ arcs, exactly one among the $E^{?}$ arcs, and any non-empty subset of the $E^{\subseteq}$ arcs.

A workflow schema can be represented in a graphical way by means of a directed acyclic graph. We represent arcs in $E^{!}$, $E^{?}$ and in $E^{\subseteq}$ with bold, dotted and dashed arcs, respectively; moreover, we draw activities in $A^{\odot}$ and in $A^{\vee}$ with bold and regular circles, respectively. Finally the ending activities are also marked with double circles.

An example of workflow that will be used throughout the paper, is shown in Figure 2. It is easy to notice that the schema actually corresponds to the explosion of the activities of the "sales ordering" process, described in the introduction[1].

A workflow schema may have associated several executions consisting of a sequence of states, where a state S is a tuple $\langle \mathit{Marked}, \mathit{Ready}, \mathit{Executed}\rangle$, with *Ready* and *Executed* subsets of activities, and *Marked* subset of the precedences.

[1] Due to space limits we do not explain this correspondence here but leave this task to the reader

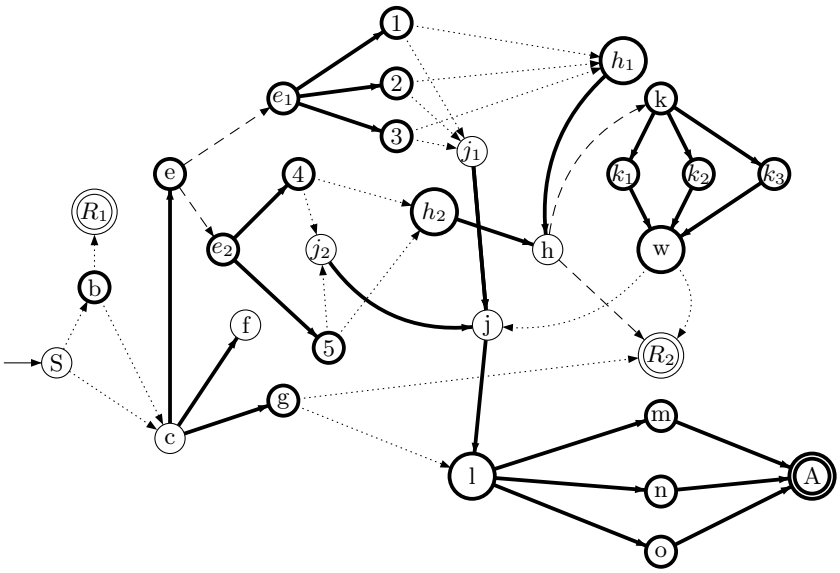

Fig. 2. An example of Workflow Schema

An execution starts with the state $S_0 = \langle \emptyset, \{a_0\}, \emptyset \rangle$. Later on, if the state at the step t is $S_t = \langle Marked_t, Ready_t, Executed_t \rangle$, then the next state S_{t+1} is one of the outcomes of a non-deterministic transition function δ, defined as follows: $\delta(S_t)$ is the set of all states $\langle Marked_t \cup Marked^{!}_{t+1} \cup Marked^{?}_{t+1} \cup Marked^{\subseteq}_{t+1}, Ready^{\vee}_{t+1} \cup Ready^{\odot}_{t+1} - Ready_t, Executed_t \cup Ready_t \rangle$, such that

- $Marked^{!}_{t+1} = \{(a,b) | \, a \in Ready_t, \, (a,b) \in E^{!}\}$, i.e., all $E^{!}$ arcs leaving ready activities are marked;
- $Marked^{?}_{t+1}$ is any maximal subset of $\{(a,b) | \, a \in Ready_t, \, (a,b) \in E^{?}\}$ s.t. there are no two distinct arcs in it leaving the same node, i.e., exactly one $E^{?}$ arc is marked by a ready activity;
- $Marked^{\subseteq}_{t+1}$ is any subset of $\{(a,b) | \, a \in Ready_t, \, (a,b) \in E^{\subseteq}\}$ s.t. for each $a \in Ready_t$ with outgoing $E^{\subseteq}$ arcs, there exists at least one $E^{\subseteq}$ arc in it;
- $Ready^{\vee}_{t+1} = \{a | a \in (A^{\vee} - Executed_t), \exists (c,a) \in Marked_t\}$, i.e., an $A^{\vee}$ activity becomes ready for execution as soon as one of its predecessor activities is completed;
- $Ready^{\odot}_{t+1} = \{a | a \in (A^{\odot} - Executed_t), \nexists (c,a) \in (E - Marked_t)\}$, i.e., an $A^{\odot}$ activity is ready for execution after all its predecessor activities are completed.

We point out that the above semantics captures the behavior of most commercial workflow products. Now we are in the position to formally define a workflow execution.

Definition 1. *Let $\mathcal{WS}$ be and δ be the transition function. An* execution *e on a workflow schema $\mathcal{WS} = \langle A, E, a_0, F \rangle$ is a sequence of states $[S_0, ..., S_k]$ such that*

1. *$S_0 = \langle \{\emptyset\}, \{a_0\}, \{\emptyset\} \rangle$,*
2. *$S_{t+1} \in \delta(S_t)$ for each $0 < t < k$, and*
3. *$Executed_k \cap F \neq \emptyset$ or $Ready_k = \emptyset$.* □

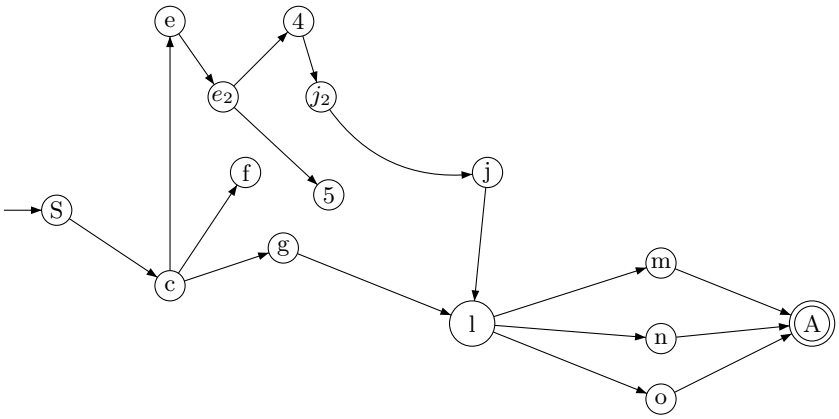

Fig. 3. An instance of the workflow schema in Fig. 2

Note that the case $Executed_k \cap F = \emptyset$ and $Ready_k = \emptyset$ (in condition 3) corresponds to an abnormal execution which does not reach a final state. As shown next, checking whether a workflow schema admits a successful execution is intractable – the hardness of the problem depends on the exclusive choices for marking the arcs in $E^?$.

Proposition 1. *Let $\mathcal{WS} = \langle A, E, a_0, F\rangle$ be a workflow schema. Then, deciding whether there exists an execution that reaches a final state (i.e., $Executed_k \cap F \neq \emptyset$) is* **NP***-complete, but the problem becomes* **P***-complete if $E^? = \emptyset$.*

Proof (sketch). Membership in **NP** is trivial. For the hardness, recall that, given a boolean formula Φ over variables $X_1, ..., X_m$ the problem of deciding whether it is satisfiable is **NP**-complete [15]. We can define a workflow schema such that each variable is associated to an activity that has two outgoing ?-arcs. Let $X_i(t)$ and $X_i(f)$ be the activities that each X_i can activate, by means of these arcs; indeed, the first step of any execution encodes a truth assignment for the variables. Finally we can define a boolean circuit, whose size is polynomial in m, that verifies whether the assignment is correct. The output gate of the circuit is the unique final state, say s, of the workflow. By construction, Φ is satisfiable if and only if there exists an execution that reaches s. On the other hand, if $E^? = \emptyset$ the problem turns to be equivalent to the graph accessibility problem for a boolean circuit. □

Workflow executions can be seen as connected subgraphs of the workflow schema and will be called instances under this graphical perspective.

Definition 2. *Let $\mathcal{WS} = \langle A, E, a_0, F\rangle$ be a workflow schema and $e = [S_0, ..., S_k]$ be an execution. Then, the* instance *associated to e is the graph $I_e = \langle A_e, E_e, a_0, F_e\rangle$, where $A_e = \cup_{t=1,k} Executed_t$, $E_e = Marked_t$ and $F_e = A_e \cap F$.* □

An instance of the workflow of Fig. 2 is reported in Fig. 3. In the following, we denote the set of all instances of a workflow schema $\mathcal{WS}$ by $\mathcal{I}(\mathcal{WS})$.

Deciding whether a subgraph is indeed an instance of $\mathcal{WS}$ is tractable although deciding the existence of an instance (i.e., whether $\mathcal{I}(\mathcal{WS}) \neq \emptyset$) is not because of Proposition 1.

Proposition 2. *Let $\mathcal{WS} = \langle A, E, a_0, F\rangle$ be a workflow schema and I be a subgraph of $\mathcal{WS}$. Then, deciding whether I is an instance of $\mathcal{WS}$ can be done in polynomial time in the size of E.*

Proof (sketch). We can simply construct the sequence of states corresponding to I by traversing the subgraph I starting from the initial node and by applying in a constructive way the function δ using all arcs in I as marked. Clearly the algorithm is polynomial in the size of E. □

3 Mining frequent patterns

In this section we present an algorithm for mining frequent patterns (i.e., subgraphs) in workflow instances. It is important to remark here that the task of discovering frequent patterns is computationally expensive. Indeed, even the problems of deciding whether an arc is included in some instance or whether a pair of nodes always occur together in every instance are not tractable.

Proposition 3. *Let $\mathcal{WS} = \langle A, E, a_0, F\rangle$ be a workflow schema. Then,*
1. *given an arc $(a, b) \in E$, deciding whether there exists an instance $I \in \mathcal{I}(\mathcal{WS})$ such that $(a, b) \in I$ is* **NP***-complete;*
2. *given two nodes a and b, deciding whether, for each $I \in \mathcal{I}(\mathcal{WS})$ a is in I implies b is in I as well is* co-**NP***-complete.* □

Let us now define the type of patterns we want to discover. Throughout the section, we assume that a workflow schema $\mathcal{WS} = \langle A, E, a_0, F\rangle$ and a set of instances $\mathcal{F} = \{I_1, ..., I_n\}$ are given. Let $2^{\mathcal{WS}}$ denote the family of all the subsets of the graph $\langle A, E\rangle$.

Definition 3. *A graph $p = \langle A_p, E_p\rangle \in 2^{\mathcal{WS}}$ is a* $\mathcal{F}$-pattern *(cf. $\mathcal{F} \models p$) if there exists $I = \langle A_I, E_I\rangle \in \mathcal{F}$ such that $A_p \subseteq A_I$ and p is the subgraph of I induced by the nodes in A_p. In the case $\mathcal{F} = \mathcal{I}(\mathcal{WS})$, the subgraph is simply said to be a* pattern. □

Let $supp(p) = |\{I | \{I\} \models p \wedge I \in \mathcal{F}\}|/|\mathcal{F}|$, be the support of a $\mathcal{F}$-pattern p. Then, given a real number *minSupp*, the problem we address consists in finding all the frequent $\mathcal{F}$-patterns, i.e. all the $\mathcal{F}$-patterns whose support is greater than *minSupp*. A frequent $\mathcal{F}$-pattern will be simply called $\mathcal{F}$-frequent.

In order to reduce the number of patterns to generate, throughout the rest of the paper, we shall only consider $\mathcal{F}$-patterns that satisfy two additional constraints: *connectivity* and *deterministic closure*. As we shall see next, these constraints do not actually reduce the generality of our approach.

The first restriction is that the undirected version of an $\mathcal{F}$-pattern must be *connected*. Indeed, the extension of the mining algorithm to cover disconnected $\mathcal{F}$-patterns can be trivially tackled by observing that any $\mathcal{F}$-pattern is frequent if and only if each of its connected components is frequent as well. Hence, the computation of frequent general patterns can be done by simply combining frequent connected components.

The second restriction is formalized next.

Definition 4. *Given a graph $p = \langle A_p, E_p \rangle \in 2^{\mathcal{WS}}$, the deterministic closure of p (cf. ws-closure(p)) is inductively defined as the graph $p' = \langle A_{p'}, E_{p'} \rangle$ such that:*

i) $A_p \subseteq A_{p'}$, *and* $E_p \subseteq E_{p'}$ *(basis of induction),*
ii) $a \in A_{p'} \cap A^{\odot} \wedge (b, a) \in E$, *implies* $(b, a) \in E_{p'}$ *and* $b \in A_{p'}$,
iii) $a \in A_{p'} \wedge (a, b) \in E^{!}$ *implies* $(a, b) \in E_{p'}$ *and* $b \in A_{p'}$.

A graph p such that $p =$ ws-closure(p) is said ws-closed. □

The above definition can be used to introduce a third notion of pattern which only depends on the structure of the workflow schema, rather than on the instances $\mathcal{F}$ or $\mathcal{I}(\mathcal{WS})$. The need of this weaker notion will be clear in a while.

Definition 5. *A* weak pattern, *or simply w-pattern, is a* ws-closed *graph $p \in 2^{\mathcal{WS}}$.* □

The following proposition characterizes the complexity of recognition for the three notions of pattern; in particular, it states that testing whether a graph is a w-pattern can be done very efficiently in deterministic logarithmic space on the size of the graph $\mathcal{WS}$ rather than on the size of $2^{\mathcal{WS}}$ as it in general happens for $\mathcal{F}$-patterns.

Proposition 4. *Let $p \in 2^{\mathcal{WS}}$. Then*

1. *deciding whether p is an $\mathcal{F}$-pattern is feasible in polynomial time in the size of $\mathcal{F}$,*
2. *deciding whether p is a pattern is* **NP***-complete (w.r.t. the input $\mathcal{WS}$).*
3. *deciding whether p is a w-pattern is in* **L** *(w.r.t. the input $\mathcal{WS}$).*

Proof (sketch).

1. By definition of $\mathcal{F}$-pattern, we can simply test if p is a subgraph of any instance.
2. The problem is in **NP** as we can guess a subgraph I, by choosing the arcs in $E^{?}$ and in $E^{\subseteq}$ to activate. Then, from Proposition 2 we can check in polynomial time that I is an instance; finally, deciding whether p is a subgraph of I, can be done in polynomial time.
   The hardness follows from Proposition 1; indeed, we can assume the pattern to be formed by a single activity, and observe that deciding if it can be executed in some execution is **NP**-hard.
3. It is easy to see that we can construct a Turing machine that, given a workflow schema and a graph p encoded into the input tapes, can decide in deterministic logarithmic space whether p is a w-pattern. In fact, we can encode both schema and graph by fixing an arbitrary order on the activities, and we can verify properties i), ii), and iii) in Definition 5, by accessing each arc of p and $\mathcal{WS}$, by means of two counters. It is obvious that such an encoding requires logarithmic space. □

It turns out that the notion of weak pattern is the most appropriate from the computational point of view. Moreover, working with w-patterns rather than $\mathcal{F}$-patterns is not an actual limitation.

Proposition 5. *Let p be a frequent $\mathcal{F}$-pattern. Then*

1. *ws-closure(p) is both a weak pattern and a frequent $\mathcal{F}$-pattern;*
2. *each weak pattern $p' \subseteq p$ is a frequent $\mathcal{F}$-pattern as well.*

Proof (sketch). In order to prove property 1, we observe that for each $I \in \mathcal{F}$ with $\{I\} \models p$, we have $\{I\} \models$ *ws-closure*(p). Indeed, if p is not a weak pattern, then according to Definition 4 there exist $a \in A_p$ such that one of the following cases occur:

- $a \in A^{\odot}$ and there exists an edge $(b, a) \in E - E_p$;
- there exists an edge $(a, c) \in E^! - E_p$

By Definition 1, each instance $I \in \mathcal{F}$ containing a, must contain b, c and $(b, a), (a, c)$ as well. As a consequence, *ws-closure*(p) is frequent as well.
In order to prove property 2, it suffices to see that if there exists an unfrequent $p' \subseteq p$, then it should contain at least either an unfrequent node a or an unfrequent edge (a, b). But this is a contradiction, since both a and (a, b) belong to p as well. □

We stress that a weak pattern is not necessarily an $\mathcal{F}$-pattern or even a pattern. As shown in the next section, we shall use weak patterns in our mining algorithm to optimize the searching space but we eventually check whether they are frequent $\mathcal{F}$-patterns.

4 The algorithm for mining frequent patterns

The algorithm we propose for mining frequent $\mathcal{F}$-patterns, uses a levelwise theory, which consist in constructing frequent weak patterns, by starting from frequent "elementary" weak patterns, defined below, and by extending each frequent weak pattern using two basic operations: adding a frequent arc and merging with another frequent elementary weak pattern. As we shall show next, the correctness follows from the results of Proposition 5, and from the observation that the space of all connected weak patterns constitutes a lower semi-lattice, with a particular precedence relation $\prec$, defined next.

The elementary weak patterns, from which we start the construction of frequent patterns, are obtained as the ws-closures of the single nodes.

Definition 6. *Let $\mathcal{WS} = \langle A, E \rangle$ be a workflow schema. Then, for each $a \in A$, the graph ws-closure$(\langle \{a\}, \{\} \rangle)$ is called an* elementary *weak pattern (cf. ew-pattern).* □

The set of all *ew*-patterns is denoted by EW. Moreover, let p be a weak pattern, then EW_p denotes the set of the elementary weak patterns contained in p. Note that given an *ew*-pattern e, EW_e is not necessarily a singleton, for it may contain other *ew*-patterns.

Given a set $E' \subseteq \mathrm{EW}$, $Compl(E') = \mathrm{EW} - \bigcup_{e \in E'} \mathrm{EW}_e$ contains all elementary patterns which are neither in E' nor contained in some element of E'.

Let us now introduce the relation of precedence $\prec$ among connected weak patterns. (Observe that the empty graph, denoted by $\perp$, is a connected weak pattern as well.) Given two connected w-patterns, say p and p', $p \prec p'$ if and only if:

a) $A_p = A_{p'}$ and $E_{p'} = E_p \cup \{(a,b)\}$, where $(a,b) \in E^{\subseteq} \cup E^{?}$ (i.e., p' is obtained from p by adding an arc), or
b) there exists $p'' \in Compl(\mathrm{EW}_p)$ such that $p' = p \cup p'' \cup X$, where X is empty if p and p'' are connected or otherwise $X = \{q\}$ and $q \in E^{\subseteq} \cup E^{?}$ connects p to p'' (i.e., p' is obtained from p by adding an elementary weak pattern and possibly a connecting arc).

Note that for each $e \in \mathrm{EW}$, we have $\perp \prec e$.

The following result states that all the connected weak patterns of a given workflow schema, can be constructed by means of a chain over the $\prec$ relation.

Lemma 1. *Let p be a connected w-pattern. Then, there exists a chain of connected w-patterns, such that $\perp \prec p_1 \prec ... \prec p_n = p$.*

Proof (sketch). We prove this by induction on the size of p, $|p| = |A_p| + |E_p|$. The base case, i.e. $p \in \perp$, is trivial. For the case $p \notin \mathrm{EW}$, assume that for each weak pattern p', such that $|p'| < |p|$ there exists a chain $\perp \prec q_1 \prec ... \prec q_m = p'$.

Two situations may occur:

1. $\exists (a,b) \in E_p \cap (E^{\subseteq} \cup E^{?})$, such that (a,b) does not belong to any elementary pattern contained in p, and the graph p' obtained from p by deleting such arc ($p' = \langle A_p, E_p - \{(a,b)\}\rangle$) is connected. In such a case, p' is a weak pattern, with $p' \prec p$. Hence, by induction, $\perp \prec q_1 \prec ... \prec q_m \prec p$. The theorem follows for $n = m$ and $p_1 = q_1, ..., p_n = q_n$.
2. for each $(a,b) \in E_p \cap (E^{\subseteq} \cup E^{?})$, such that (a,b) does not belong to any elementary pattern contained in p, the graph $p' = \langle A_p, E_p - \{(a,b)\}\rangle$ is not connected. Two subcases can be further devised:
   (a) there exists an elementary weak pattern $e \in \mathrm{EW}_p$, which is connected to the graph $p - e$ by means of exactly one arc in $E^{\subseteq} \cup E^{?}$; that is $e \in Compl(\mathrm{EW}_{p-e})$, and, hence $(p - e) \prec p$, and the theorem follows by induction, otherwise
   (b) elementary patterns are not connected by means of arcs in $E^{\subseteq} \cup E^{?}$. In this case, let $ep_0, ep_1, ..., ep_m$ be the elementary patterns contained in p, and $q = (p - ep_0) \cup ep_1 \cup ... \cup ep_m$ the weak pattern obtained from p by deleting edges and nodes in ep_0 which do no occur in any other ep_i, with $0 < i \leq m$. By construction $ep_0 \in Compl(q)$, and hence $q \prec p$. As in the other case, since $|q| < |p|$, by induction there exists a chain of weak patterns $\perp \prec q_1 \prec ... \prec q_m = q$. □

It turns out that the space of all connected weak patterns is a lower semi-lattice w.r.t. the precedence relation $\prec$. The algorithm w-find, reported in Figure 4, exploits an apriori-like exploration of this lower semi-lattice.

At each stage, the computation of L_{k+1} (steps 5-14) is carried out by extending any pattern p generated at the previous stage ($p \in L_k$), in two ways:

- by adding frequent edges in $E^{\subseteq}$ or $E^{?}$ (*addFrequentArc* function), and
- by adding an elementary weak patterns (*addEWFrequentPattern* function).

The properties of the algorithm are reported in the following lemma.

Lemma 2. *In the algorithm w-find, reported in Figure 4, the following propositions hold:*

a. L_0 *contains a set of frequent connected* $\mathcal{F}$*-pattern;*
b. addFrequentArc add to U *connected patterns, which are not necessary* $\mathcal{F}$*-patterns;*
c. addFrequentEWPattern add to U *connected w-patterns, which are not necessary patterns;*
d. L_{k+1} *contains only frequent connected* $\mathcal{F}$*-patterns.* □

Corollary 1. *(Soundness) The set* R *computed by the algorithm of Figure 4 contains only frequent connected* $\mathcal{F}$*-patterns.* □

Proposition 6. *(Completeness) The algorithm of Figure 4 terminates and computes all the frequent connected weak patterns. In particular, let* R *be the set of* $\mathcal{F}$*-patterns computed by the algorithm. Then, for each frequent connected weak pattern* p*, we have* $p \in R$*.*

Proof (sketch). The algorithm w-find computes all the element in the lower semi-lattice induced by the operator $\prec$ over w-patterns. The completeness follows from Lemma 1, that states that any weak pattern is represented by a chain in this lower semi-lattice, and by the observation that we also prune the chains that will lead to unfrequent pattern. The latter is done by replacing the function *addEWPattern* in the definition of the relation $\prec$ with *addFrequentEWPattern*. □

5 Experiments

In this section we study the behavior of the algorithm, by evaluating both its performances and its scalability. As shown in the previous section, the algorithm is sound and complete w.r.t. the set of frequent w-patterns. Nevertheless, the number of candidate w-patterns generated could be prohibitively high, thus making the algorithm unfeasible on complex workflow schemas.

As already mentioned in Section 1, several existing techniques for computing frequent itemsets can be viable solutions to the problem of mining frequent instances in a workflow, provided that suitable representations of instances is exploited. Thus, we compare the *w-find* algorithm with some of them. In particular, we consider an implementation of the *Apriori* algorithm which computes frequent itemsets of edges in $E^{\subseteq} \cup E^{?}$, and next connects such arcs with arcs in $E^{!}$.

Input: A workflow Graph $\mathcal{WS}$, a set $\mathcal{F} = \{I_1, \ldots, I_N\}$ of instances of $\mathcal{WS}$.
Output: A set of frequent $\mathcal{F}$-patterns.
Method: Perform the following steps:

```
1   L_0 := {e | e ∈ EW, e is frequent w.r.t. F};      //see Lemma 2.a
2   k := 0, R := L_0;
3   FrequentArcs := {(a,b) | (a,b) ∈ (E^⊆ ∪ E^?), ⟨{a,b},{(a,b)}⟩ is frequent w.r.t. F};
4   E_f^⊆ := E^⊆ ∩ FrequentArcs, E_f^? := E^? ∩ FrequentArcs;
5   repeat
6     U := ∅;
7     forall p ∈ L_k do begin
8       U := U ∪ addFrequentArc(p);      //see Lemma 2.b
9       forall e ∈ Compl(EW_p) ∩ L_0 do
10        U := U ∪ addFrequentEWPattern(p, e);      //see Lemma 2.c
11    end
12    L_{k+1} := {p | p ∈ U, p is frequent w.r.t. F};      //see Lemma 2.d
13    R := R ∪ L_{k+1};
14  until L_{k+1} = ∅;
15  return R;
```

Function *addFrequentEWPattern*$(p = \langle A_p, E_p\rangle, e = \langle A_e, E_e\rangle)$: **w-pattern**;

```
p' := ⟨A_p ∪ A_e, E_p ∪ E_e⟩;
if p' is connected, then return p' else return addFrequentConnection(p, e);
```

Function *addFrequentConnection*$(p = \langle A_p, E_p\rangle, e = \langle A_e, E_e\rangle)$: **w-pattern**;

```
S := ∅
forall frequent (a,b) ∈ (E_f^⊆ ∪ E_f^?) − E_p s.t. (a ∈ A_p, b ∈ A_e) ∨ (a ∈ A_e, b ∈ A_p) do begin
  p' := ⟨A_p, E_p ∪ (a,b)⟩;
  if WS ⊨ p' then S := S ∪ {p'};
end
return S
```

Function *addFrequentArc*$(p = \langle A_p, E_p\rangle)$: **pattern**;

```
S := ∅
forall frequent (a,b) ∈ (E_f^⊆ ∪ E_f^?) − E_p s.t. a ∈ A_p, b ∈ A_p do begin
  p' := ⟨A_p, E_p ∪ (a,b)⟩
  if WS ⊨ p' then S := S ∪ {p'};
end
return S
```

Fig. 4. Algorithm w-find($\mathcal{F}$,$\mathcal{WS}$)

A possible further approach to consider is the *WARMR* algorithm devised in [13], that allows an explicit formalization of domain knowledge (like, for example, the connectivity information provided by the workflow schema) which can be directly exploited by the algorithm. However, due to space limitations, we omit the results of such comparison here.

In our experiments, we mainly use synthetic data. Synthetic data generation can be tuned according to: i) the size of $\mathcal{WS}$, ii) the size of $\mathcal{F}$, iii) the size $|L|$ of the frequent weak patterns in $\mathcal{F}$, and iv) the probability $p_{\subseteq}$ of choosing a $\subseteq$-arc. The ideas adopted in building the generator for synthetic data are essentially inspired by [3].

In a first set of experiments, we consider some fixed workflow schemas, and generate synthesized workflow instances, by simulating a number of execution. In particular, for each node a in a ready state, exactly one of the edges $(a, b) \in E^?$

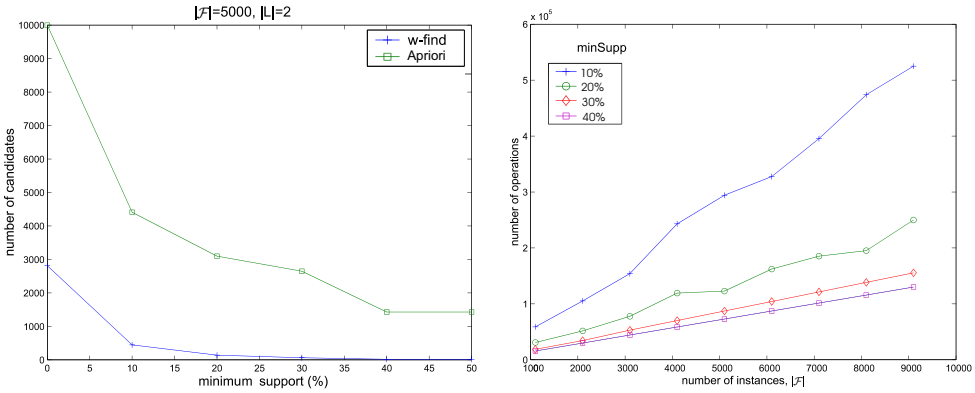

Fig. 5. Left: Number of candidates w.r.t. *minSupp*. Right: Number of matching operations w.r.t. $|\mathcal{F}|$.

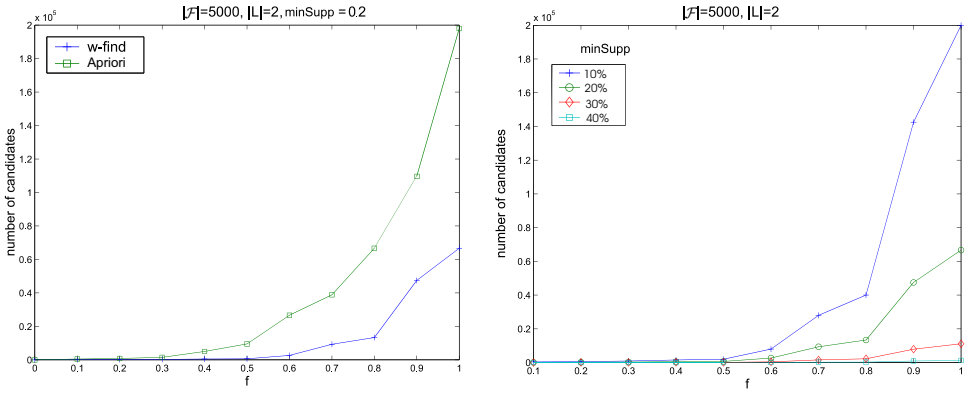

Fig. 6. Number of candidates w.r.t. f, for randomly generated workflow schemas. Left: comparison with a-priori. Right: w-find performances for different *minSupp* values.

and a subset of the edges $(a, b) \in E^{\subseteq}$ are randomly chosen. The number of $\subseteq$-arcs is chosen by picking from a binomial distribution with mean $p_{\subseteq}$. Frequent instances are forced into the system by replicating some instances (in which some variations were randomly performed) according to $|L|$.

We perform several experiments comparing the performance of *Apriori* and *w-find* on increasing values of $|\mathcal{F}|$ and *minSupp*. For a dataset of instances generated w.r.t. the workflow schema of Figure 2, the comparison is reported in Figure 5. As expected, *w-find* outperforms *Apriori* of an order of magnitude. This is mainly due to the fact that, contrarily to *w-find*, in the *Apriori* implementation arcs in $E^{?} \cup E^{\subseteq}$ are combined without taking into account the information provided by the workflow schema.

Figure 5(on the right) reports the number of operations (matching of a pattern with an instance), for increasing values of $\mathcal{F}$. The figure shows that the algorithm scales linearly in the size of the input (for different supports).

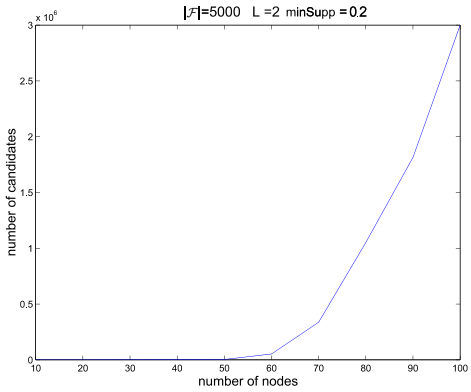

Fig. 7. Number of candidates w.r.t. the number of nodes in the workflow schema.

In a second set of experiments, we randomly generate the workflow schemas to test the efficiency of the approach w.r.t. the structure of the workflow. To this purpose, we fix $|\mathcal{F}|$ and generate workflow instances according to the randomly generated schema. The actual number of nodes and arcs (i.e., each of $|A^{\odot}|$, $|A^{\vee}|$, $|E^{!}|$, $|E^{?}|$ and $|E^{\subseteq}|$), is chosen by picking from a Poisson distribution with fixed mean value.

In order to evaluate the contribution of the complexity of workflow schemas, we exploit the factor $f = \frac{|E_?|+|E_\subseteq|}{|E_?|+|E_\subseteq|+|E_!|}$, which represents the degree of potentional nondeterminism within a workflow schema. Intuitively, workflow schemas exhibiting $f \simeq 0$ produce instances with a small number of candidate w-patterns. Conversely, workflow schemas exhibiting $f \simeq 1$ produce instances with a huge number of candidate w-patterns. Figure 5 shows the behavior of both *Apriori* and *w-find* when f ranges between 0 (no nondeterminsm) and 1 (full nondeterminism). Again, *Apriori* is outperformed by *w-find*, even though for small values of f both the algorithms produce a small number of candidates.

Finally, Figure 7 reports the correlation between the number of candidate w-patterns and the number of nodes in the workflow schema.

6 Conclusions

In this paper we presented an efficient algorithm for mining frequent instances of workflow schemas. The main motivation for this work was aimed at providing facilities for the human system administrator to monitor the actual behavior of the workflow system in order to predict the "most probable" workflow executions. In this context, the use of mining techniques is justified by the fact that even "simple" reachability problems become intractable.

The proposed algorithm was shown to efficiently explore the search space; hence, it can be exploited as an effective mean for investigating some inherent properties of the executions of a given workflow schema.

We conclude by mentioning some directions of future research. The proposed model is essentially a *propositional* model, for it assumes a simplification of the workflow schema in which many real-life details are omitted. However, we believe that the model can be easily updated to cope with more complex constraints, such as time constraints, pre-conditions and post-conditions, and rules for exception handling.

References

1. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In*Proc. 6th Int. Conf. on Extending Database Technology (EDBT)*, pages 469–483, 1998.
2. R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. ACM Conf. on Management of Data (SIGMOD93)*, pages 207–216, 1993.
3. R. Agrawal, and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases In *Proc. 20th Int. Conf. on Very Large Data Bases(VLDB94)*, pages 487–499,1994.
4. R. Agrawal, and R. Srikant. Mining Sequential Patterns. In *Proc. 11th Int. Conf. on Data Engineering (ICDE95)*, pages 3–14, 1995.
5. R. J. Bayardo Jr. Efficiently Mining Long Patterns from Databases. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, pages 85–93, 1998.
6. A. Bonner. Workflow, Transactions, and Datalog. In *Proc. of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 294–305, 1999.
7. F. Casati, and G. Pozzi. Modeling Exceptional Behaviors in Commercial Workflow Management Systems. In*Proc. 4th Int. Conf. on Cooperative Information Systems (IFCIS99)*, pages 127–138, 1999.
8. Q. Chen, and U. Dayal. Failure Handling for Transaction Hierarchies. In *Proc. 13th Int. Conf. on Data Engineering (ICDE97)*, pages 245–254, 1997.
9. E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding Interesting Associations without Support Pruning. *IEEE Transactions on Knowledge and Data Engineering*, 13(1), pages 64–78, 2001.
10. J. E. Cook, and A. L. Wolf. Automating Process Discovery Through Event-Data Analysis. In *Proc. 17th Int. Conf. on Software Engineering (ICSE95)*, pages 73–82, 1995.
11. H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic Based Modeling and Analysis of Workflows. In*Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 25–33, 1998.
12. U. Dayal, M. Hsu, and R. Ladin. Business Process Coordination: State of the Art, Trends, and Open Issues. In*Proc. 27th Int. Conf. on Very Large Data Bases (VLBDB01)*, pages 3-13, 2001.
13. L. Dehaspe, and H. Toivonen. Discovery of Frequent DATALOG Patterns. *Data Mining and Knowledge Discovery*, 3(1),pages 7–36, 1999.
14. T. Feder, P. Hell, S. Klein, and R. Motwani. Complexity of Graph Partition Problems. *STOC*, pages 464–472, 1999
15. M. R. Garey, and D. S. Johnson. Computers and Intractability. A Guide to the Theory of **NP**-completeness, Freeman and Comp., NY, USA, 1979.
16. D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2), pages 119–153, 1995.
17. M. Gillmann, W. Wonner, and G. Weikum. Workflow Management with Service Quality Guarantees. In *Proc. ACM Conf. on Management of Data (SIGMOD02)* , 2002.
18. P. Grefen, J. Vonk, and P. M. G. Apers. Global transaction support for workflow management systems: from formal specification to practical implementation. *VLDB Journal* 10(4), pages 316–333, 2001.

19. G. Lee, K. L. Lee, and A. L. P. Chen. Efficient Graph-Based Algorithms for Discovering and Maintaining Association Rules in Large Databases. *Knowledge and Information Systems* 3(3), pages 338–355, 2001.
20. M. Kamath, and K. Ramamritham. Failure handling and coordinated execution of concurrent workflows. In*Proc. 14th Int. Conf. on Data Engineering (ICDE98)*, pages 334–341, 1998.
21. P. Koksal, S. N. Arpinar, and A. Dogac. Workflow History Management. *SIGMOD Record*, 27(1), pages 67–75, 1998.
22. A. Inokuchi, T. Washio, and H. Motoda. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In *Proc. 4th European Conf. on Principles of Data Mining and Knowledge Discovery*, pages 13–23, 2000.
23. T. Miyahara and others. Discovery of Frequent Tag Tree Patterns in Semistructured Web Documents. In *Proc 6th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining*, pages 356–367, 2002.
24. P. Muth, J. Weienfels, M. Gillmann, and G. Weikum. Integrating Light-Weight Workflow Management Systems within Existing Business Environments. In *Proc. 15th Int. Conf. on Data Engineering*, pages 286–293, 1999.
25. W. M. P. van der Aalst. The Application of Petri Nets to Worflow Management. *Journal of Circuits, Systems, and Computers*, 8(1), pages 21–66, 1998.
26. W. M. P. van der Aalst, and K. M. van Hee. Workflow Management: Models, Methods, and Systems. *MIT Press*, 2002.
27. D. Wodtke, and G. Weikum. A Formal Foundation for Distributed Workflow Execution Based on State Charts. In *Proc. 6th Int. Conf. on Database Theory (ICDT97)*, pages 230–246, 1997.
28. The Workflow Management Coalition, *http://www.wfmc.org/*.
29. M. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning* 42(1/2), pages 31–60, 2001.
30. M. Zaki. Efficiently Mining Frequent Trees in a Forest. In *Proc. 8th Int. Conf. On Knowledge Discovery and Data Mining (SIGKDD02)*, 2002. to appear.