Mining Constrained Graphs: The Case of Workflow Systems

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

Department of Mathematics[1], University of Calabria, Italy
ICAR, CNR[2], Italy
ggreco@mat.unical.it,{guzzo,manco,pontieri,sacca}@icar.cnr.it

Abstract. Constrained graphs are directed graphs describing the control flow of processes models. In such graphs, nodes represent activities involved in the process, and edges the precedence relationship among such activities. Typically, nodes and edges can specify some constraints, which control the interaction among the activities. Faced with the above features constrained graphs are widely used in the modelling and analysis of Workflow processes. In this paper we overview two mining problems related to the analysis of constrained graphs, namely the analysis of frequent patterns of execution, and the induction of a constrained graph from a set of execution traces. We discuss some complexity aspects related to the problem of reasoning and mining on constrained graphs, and overview two algorithms for the mentioned problems.

1 Introduction

Graph-based models have been widely used in several contexts as an intuitive and yet formal way of representing several kinds of data, like, e.g., web documents, chemical compounds, process models, behavioral patterns. Graph structures can be exploited both for representing a given application domain, and for modelling relationships between the involved objects, by means of constraints over the underlying graph structure. In this perspective, constrained graphs are a powerful means for representing many classes of applications requiring complex modelling structures, and can profitably support reasoning and mining tasks.

As an example, constrained graphs are used in the modelling of workflow processes. In Workflow Management Systems, the structure of a process is commonly represented by a *control graph*, where nodes correspond to the involved tasks while edges represent the potential flow of work, i.e., the precedence relationships defined among the tasks. An example constrained (control flow) graph is shown in Figure 1, for modelling a toy (*OrderManagement*) process for handling customers' orders. Several constraints can be specified over the control graph, expressing, e.g., conditions for the occurrence of some nodes in the graph in any execution. For example, some constraints in figure are the following: (i) exactly one of the outgoing edges of node b must appear in any (execution) instance including b, and (ii) if node l occurs in an instance, both its incoming edges must appear as well in the same instance.

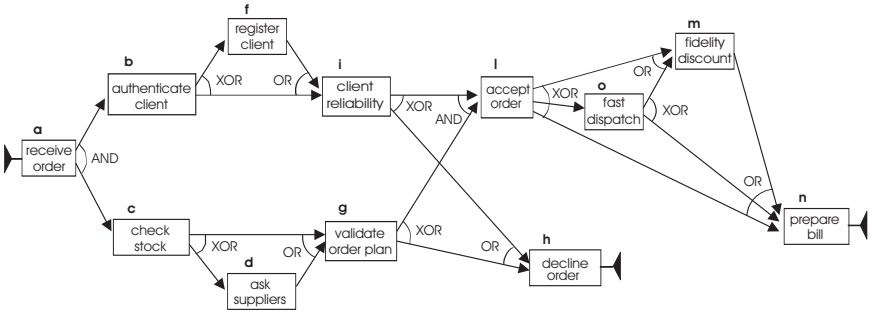
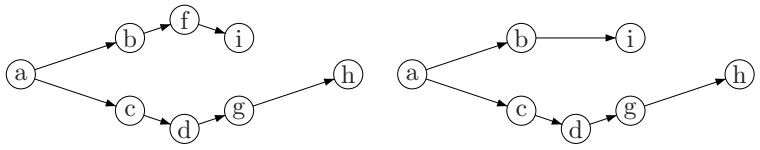


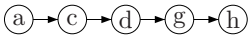
Fig. 1. Control flow graph for *OrderManagement* process.

Thus, constraint graphs model the behavior of generic processes, whose executions can be traced and stored into database structures. In this context, a challenging research direction is the definition of both suitable pattern domains and mining techniques for the underlying data. Formally, the problem of mining constrained graphs can be formulated as follows. We are given a graph schema $\mathcal{WS}$ (i.e., a graph in which both nodes and edges must satisfy some specified constraints). An instance of a graph schema is any subgraph of $\mathcal{WS}$ which satisfies the constraints. An example is the subgraph containing nodes `a`, `c`, `b`, `i`, `g` and `h` in Figure 1. The subgraph describes the processing of an order which is declined due to a failure in the validation of the order plan. Hence, for a given pattern language $\mathcal{L}$, a set of instances $\mathcal{F}$ and a boolean inductive query Q, we aim at finding the inductive theory $\mathcal{T}h(\mathcal{F}, \mathcal{L}, Q) = \{p \in \mathcal{L} | Q(\mathcal{F}, p)\}$.

In this paper we describe two different pattern domains. A first case, formerly studied in [13], raises when patterns are subgraphs of the instances. Here, inductive queries can be used to formulate frequent pattern discovery problems. In particular, one can be interested in finding the discriminant factors which characterize a desired workflow configuration: essentially, this means finding frequent patterns containing some given portions of the workflow schema (i.e., finding all the patterns p satisfying $Q_f(\mathcal{F}, p) \wedge Q_1(\mathcal{F}, p) \wedge \ldots \wedge Q_n(\mathcal{F}, p)$, where $Q_f(\mathcal{F}, p)$ is true if p is frequent w.r.t. $\mathcal{F}$, and $Q_i(\mathcal{F}, p)$ is true if p contains a given subgraph g_i). For example, one could be interested in knowing which activities, among the ones described in Figure 1, are included within the frequent paths of execution which also include node `h` (representing the rejection of an order). Consider, e.g., the following toy instances:



Notice that both instances are subgraphs of the schema shown in Figure 1, and satisfy the constraints specified there. In addition, both the instances comply with the requirement of containing node `h`. Now, the subgraph



frequently occurs in both instances, and is characterized by the co-occurrence of activities c,d, g. Essentially, this means that in the modelled *OrderManagement* process, the rejection of an order is often characterized by the lack of availability of supplies. In a business intelligence perspective, this would require a better strategy for managing the store.

The challenge in the exemplified pattern domain is the generation of the desired patterns by a smart exploration of the search space, which benefits from the presence of schema constraints.

Another interesting situation occurs when the schema $\mathcal{WS}$ is not known a priori, although some instances are available and can be examined. In such a case, inductive queries can be used to formulate and solve the mining problem of inducing the schema. An example in this setting is the *Process mining problem* [12], where data collected during the enactment of a process is exploited to reconstruct the structure of the process. In more detail, we are given a set $\mathcal{F}$ of instances, and a language of patterns $\mathcal{L}$, modelling graph schemata. A boolean inductive query $Q_P(\mathcal{F}, p)$ is satisfied whenever p is a process model for $\mathcal{F}$, i.e., whenever each instance in $\mathcal{F}$ is also an instance of the graph schema represented by p. Again, many variants of the Q_P problem can be defined, by requiring, e.g., that p contains a given subgraph, or that satisfies a given constraint.

The main challenge here is devising efficient techniques to produce accurate and yet intuitive process models. As a matter of fact, since many models, in principle, could support a given set $\mathcal{F}$ of instances, a criterion could be devised to single the ones with satisfactory modelling features. For example, it is reasonable to require that, besides representing all the instances in $\mathcal{F}$, the discovered model has a limited description size and admits a minimal number of "spurious" execution paths, which do not have any correspondence in $\mathcal{F}$.

Objectives. In this paper, we elaborate the above described issues, by defining a formal model for mining constrained graphs, and by illustrating some efficient techniques for extracting patterns from graph-based data. In more detail, the contribution of the paper can be summarized as follows. In section 2, we introduce a formal framework for representing graph-based data, and classes of constraints over such data. We discuss some complexity results in reasoning on constrained graphs. Next, we state two mining problems, namely the mining of constrained subgraphs in section 3, and the induction of graph-based models in section 4. We discuss some approaches to the solution of the proposed mining problems, and show that they can be effectively exploited to support reasoning on constrained graphs. Throughout the paper, we exploit workflow management as a relevant application context for constrained graphs, and show that the proposed solutions suitably apply to the problem of workflow modelling.

Related Work. Mining workflow pattern is emerging as a novel field of research, promising interesting research issues and raising challenges in workflow applications. Since it is a pioneering study, techniques for workflow mining can be

preliminary compared with several approaches proposed to mine patterns for structured or sequential data [2, 14, 3, 25, 24]. Moreover, they have also a strong relation with mining of graph structured data, occurring in several practical domains such as biology, chemistry and communication networking.

Many papers on graph mining have been proposed in the last years. A first category of studies applies greedy search to find subgraph patterns [4, 33]. These approaches avoid the high complexity of the graph isomorphism problem, by mining an incomplete set of characteristic subgraphs. Conversely, a complete search for frequent subgraphs is guaranteed in WARMR, an ILP-based algorithm proposed by Dehaspe and Toivonen [9]. They formulated the problem of carcinogenesis prediction of chemical compounds with a set of grounded first order predicates representing graphs and they resolved this problem by combining ILP method with Apriori-like level wise search. Other approaches performing either level-wise search or projection methods to mine a complete set of subgraphs were proposed as well [17, 19, 31, 32].

In principle, many of the above approaches could be used to mine constrained graphs. However, the adaptation of the above mentioned methods to workflow mining is a challenging task, and it results unpractical from both the expressiveness and the efficiency viewpoint. Indeed, generation of patterns with such traditional approaches does not benefit from the exploitation of the executions' constraints imposed by the workflow schema, such as precedences among activities, synchronization and parallel executions of activities (see, e.g, [18, 29]).

In this setting, more sophisticated techniques have been successfully applied, in order to derive formal graph models from graph instances. The first example in the context of Workflow mining is in [1], where the main objective is the induction of a *directed graph model* exhibiting a limited number of control flow constructs.

Other more sophisticated approaches have been devised, relying, e.g., on the notion of grammar inference [23, 5–7], or Petri Nets [29, 27, 28, 30, 8]. Starting from workflow logs, i.e., collections of linearized graph instances, the mentioned approaches propose algorithms for inducing complex control flow constraints. Further approaches have been devised in [15, 16] and [26], where richer representation languages are adopted to discover more complex graph structures. In particular, the former approaches are devoted to the detection of redundancies in the workflow model, while the latter discover hierarchically structured workflow processes.

2 An Abstract Model for Constrained Graphs

A significant amount of research has been done in the specification of mechanisms for modelling processes; in particular, several formalisms have been proposed in the area of process modelling for software engineering (see, e.g. [10] for an overview of different proposals). The most widely adopted formalism is the *control flow graph*, in which a process is represented by a labelled directed graph whose nodes correspond to the tasks to be performed, and whose arcs de-

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

Any connected subgraph $I = \langle A_I, E_I\rangle$ of the control flow graph, such that $a_0 \in A_I$ and $A_I \cap F \neq \emptyset$ is a *potential instance* of P. In order to model restrictions on the possible instances, the description of a constrained graph is often enriched with *local* and *global* constraints, which express further relationships among the activities appearing in the control graph.

In particular, *local constraints* specify local properties of a given activity, with respect to its adjacents. For instance, possible local constraints are that an activity either can be executed only after all its predecessors are completed.

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

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

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

As for the semantics, an activity a can start as soon as at least $\texttt{IN}(a)$ of its predecessor activities have been completed. Two typical cases are: (i) if $\texttt{IN}(a) = \mathit{InDegree}(a)$ then a is an *and-join* activity, for it can be executed only after all of its predecessors are completed, and (ii) if $\texttt{IN}(a) = 1$ then a is an *or-join* activity, for it can be executed as soon as one of its predecessors is completed. Once finished, an activity a activates one non-empty subset of its outgoing arcs with cardinality between $\texttt{OUT}_{min}(a)$ and $\texttt{OUT}_{max}(a)$. If $\texttt{OUT}_{max}(a) = \mathit{OutDegree}(a)$ then a is a *full fork* and if also $\texttt{OUT}_{min}(a) = \texttt{OUT}_{max}(a)$ then a is a *deterministic fork* (also known as "and-split"), for it activates all of its successor activities. Finally, if $\texttt{OUT}_{max}(a) = 1$ then a is an *exclusive fork* (also called *xor-split* in the literature), for it activates exactly one of its outgoing arcs. Figure 1 shows an example schema containing the above mentioned constraints.

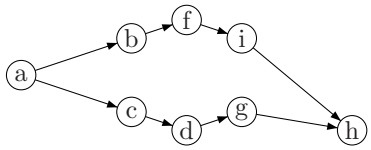
Global constraints specify relationships among not necessarily connected activities. Such constraints are richer in nature and their representation strongly depends on the particular application domain of the modelled process. Thus, they are often expressed using complex formalisms. Here, we assume that global constraints are propositional formulas expressing relationships among the nodes in A and edges in E. As an example, the constraint $f \rightarrow \neg m$ states that whenever activity f occurs, activity m cannot occur. This constraint, referred to Figure 1, has the intuitive meaning that fidelity discounts cannot be applied to new clients.

For a generic process P, a *workflow schema* for P, denoted by $\mathcal{WS}(P)$, is a tuple $\langle \mathcal{CF}(P), \mathcal{C}_L(P), \mathcal{C}_G(P)\rangle$, where $\mathcal{CF}(P)$ is the control flow graph of P, and $\mathcal{C}_L(P)$ and $\mathcal{C}_G(P)$ are sets of local and global constraints, respectively.

Given a subgraph I of $\mathcal{CF}(P)$ and a constraint c in $\mathcal{C}_L(P) \cup \mathcal{C}_G(P)$, we write $I \models c$ whenever I satisfies c in the associated semantics. Moreover, if $I \models c$ for

all c in $\mathcal{C}_L(P) \cup \mathcal{C}_G(P)$ and contains both the starting activity a_0 and a final activity in F, then I is called an *instance* of $\mathcal{WS}(P)$, denoted by $I \models \mathcal{WS}(P)$. When the process P is clear from the context, a workflow schema will be simply denoted by $\mathcal{WS} = \langle \mathcal{CF}, \mathcal{C}_L, \mathcal{C}_G \rangle$.

Example 1. The following is an example instance of the workflow process $\mathcal{WS}$ shown in Figure 1.



Notice how each node appearing within the instance satisfies the constraints specified by $\mathcal{WS}$. ◁

Checking whether a workflow schema admits a successful execution is intractable.

Proposition 1 ([13]). *Let $\mathcal{WS} = \langle \mathcal{CF}, \mathcal{C}_L, \mathcal{C}_G \rangle$ be a workflow schema. Then, deciding whether there exists an instance I is* **NP***-complete, but the problem becomes* **P***-complete if all nodes are full-forks.* □

The above proposition has a strong negative impact: we cannot statically induce relevant properties of a workflow schema. This justifies the adoption of data mining techniques, which in principle allow to dynamically induce the desired properties from the instances resulting from past executions. Precisely, we assume that each instance is properly stored by the workflow management system in the log file, which can be seen as a set $\mathcal{F} = \{I_1, ..., I_n\}$ such that $\mathcal{WS} \models I_i$, for each $1 \leq i \leq n$. In the following, we denote by $\mathcal{I}(\mathcal{WS})$ the set of all the instances of a given workflow $\mathcal{WS}$.

Deciding whether a subgraph is 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 ([13]). *Let $\mathcal{WS} = \langle \mathcal{CF}, \mathcal{C}_L, \mathcal{C}_G \rangle$ be a workflow schema and I be a subgraph of $\mathcal{CF}$. Then, deciding whether I is an instance of $\mathcal{WS}$ can be done in polynomial time in the size of E.* □

Usually, logs are stored by means of traces. A *workflow trace s over A* is a string in A^*, representing an instance . Given a trace s, we denote by $s[i]$ the *i-th* task in the corresponding sequence, and by $lenght(s)$ the length of s. The set of all the tasks in s is denoted by $tasks(s) = \bigcup_{1 \leq i \leq lenght(s)} s[i]$. Hence, a *workflow log for $\mathcal{WS}(P)$*, denoted by $\mathcal{L}_P$, is a multiset of traces: $\mathcal{L}_P = [\ s \mid s \in A^*\ ]$.

Let s be a trace in $\mathcal{L}_P$, $\mathcal{WS}$ be a workflow schema, and $I = \langle A_I, E_I \rangle$ be an instance of $\mathcal{WS}$. Then, s is *compliant with $\mathcal{WS}$ through I*, denoted by $\mathcal{WS} \models^I s$, if s is a topological sort of I, i.e., s is an ordering of the activities in A_I s.t. for

each $(a, b) \in E_I$, $i < j$ where $s[i] = a$ and $s[j] = b$. Moreover, s is *compliant with* $\mathcal{WS}$, denoted by $\mathcal{WS} \models s$, if there exists I with $\mathcal{WS} \models^I s$. Finally, a weaker notion of compliance, which does not rely on the existence of an instance I, can be defined as $\mathcal{WS} \vdash s$. The latter holds whenever the order of appearance of the activities in s is compatible with the constraints specified in $\mathcal{WS}$.

Example 2. The following table reports example log traces for the process $\mathcal{WS}$ shown in Figure 1.

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

By considering the instance I of example 1, we can observe that $\mathcal{WS} \models^I s_1$. $\triangleleft$

Proposition 3. *Let $\mathcal{WS} = \langle \mathcal{CF}, \mathcal{C}_L, \mathcal{C}_G \rangle$ be a workflow schema and s be a trace of $\mathcal{CF}$. Then, deciding whether $\mathcal{WS} \models s$ is* **NP***-complete, but deciding whether $\mathcal{WS} \vdash s$ and, given an instance I, whether $\mathcal{WS} \models^I s$ can be done in polynomial time in the size of E.*

Proof. We first show that deciding whether $\mathcal{WS} \models s$ is **NP**-complete. 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 Φ is satisfiable is **NP**-complete. W.l.o.g. assume Φ to be in conjunctive normal form. Then, we define a workflow schema $\mathcal{WS}(\Phi) = \langle \mathcal{CF}, \mathcal{C}_L, \mathcal{C}_G \rangle$, where $\mathcal{CF} = \langle A, E, a_o, \{Sat\} \rangle$, such that A consists of an initial activity a_0, of the activities X_i, TX_i, FX_i, B_i for each $0 < i \leq m$, of the activities C_j for each distinct clause j of Φ, and the activity B, and of a final state Sat. The set of local constraints $\mathcal{C}_L$ and dependencies in E is defined as follows. Let $\texttt{IN}(Sat) = n$ (where n is the number of clauses contained in Φ), and $\texttt{IN}(a) = 1$ for any other activity $a \neq a_0$. Moreover:

- For each X_i, (X_i, TX_i), (X_i, FX_i), (B_i, TX_i), (B_i, FX_i), (TX_i, B), and (FX_i, B) are in E, with constraints $\texttt{OUT}_{min}(X_i) = \texttt{OUT}_{max}(X_i) = 1$ and $\texttt{OUT}_{min}(B_i) = \texttt{OUT}_{max}(B_i) = 1$. Thus, each time either the activity X_i or B_i is executed, it is required to make a choice between its possible successors; note that in our encoding, TX_i means that X_i is **true**, while FX_i means that X_i is **false**. Finally the arcs (a_0, X_i) and (a_o, B_i) are in E, and constraints $\texttt{OUT}_{min}(a) = \texttt{OUT}_{max}(a) = m + m$ are added.
- For each C_j, we have that (C_j, Sat) is in E, with constraints $\texttt{OUT}_{min}(Sat) = \texttt{OUT}_{max}(Sat) = 1$. Moreover, we have $(TX_i, C_j) \in E$ in the case X_j appears in the clause C_j, while we have $(FX_i, C_j) \in E$ in the case X_i appears negated in the clause C_j. Finally, for each node $a \in \{TX_i, FX_i\}$, $\texttt{OUT}_{min}(a) = 1$ and $\texttt{OUT}_{max}(a) = \mathit{OutDegree}(a) - 1$.

Global constraints in $\mathcal{C}_G$ are defined as follows. For each pair of activities of the form X_i and B_i, there is a constraint stating that the arc (B_i, TX_i) (resp. (B_i, FX_i)) cannot occur in the same execution with the arc (X_i, TX_i) (resp.

(X_i, FX_i)). Moreover, for each activity of the form X_i, there is a constraint stating that arcs of the form (TX_i, C_j) (resp. (FX_i, C_j)) cannot occur in the same execution with arcs (TX_i, B) (resp. (FX_i, B)); finally, for each activity X_i, there is a constraint stating that an arc of the form (B_i, TX_i) (resp. (B_i, FX_i)) implies the activation of the arc (TX_i, B) (resp. (FX_i, B)).

Consider now a trace $s(\Phi) = a_0 B_1, ..., B_m X_1 ... X_m B C_1 ... C_m Sat$. Then, it is easy to see that $\mathcal{WS} \models s$ if and only if Φ is satisfiable.

To conclude the proof observe that (*1*) in order to decide whether $\mathcal{WS} \vdash s$ it is sufficient to tests the topological relationships locally induced by s, and that (*2*) in order to decide whether $\mathcal{WS} \models^I s$ it is sufficient to simulate the enactment of I. Both the above tasks are feasible in polynomial time. □

3 Mining Frequent Patterns

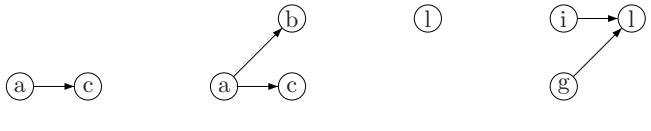
In this section we address the problem of mining connected frequent patterns (i.e., subgraphs) in workflow instances. Let us assume that a workflow schema $\mathcal{WS} = \langle \mathcal{CF}, \mathcal{C}_L, \mathcal{C}_G \rangle$ and a multiset of instances $\mathcal{F} = \{I_1, ..., I_n\}$ are given. A graph $p = \langle A_p, E_p \rangle \subseteq \mathcal{CF}$ 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 $supp(p) = |\{I \in \mathcal{F} | I \models p\}|/|\mathcal{F}|$, be the *support* of a $\mathcal{F}$-pattern p. Then, given a real number σ, we consider the following problem:

`FCPD`(σ)**:** *Frequent Connected Pattern Discovery*, i.e., finding all the connected patterns whose support is greater than σ.

A naive algorithm for mining frequent patterns can generate directly connected subgraphs, and then test in polynomial time whether it is indeed an instance of $\mathcal{WS}$. A different approach is based on the idea of reducing the number of patterns to generate. To achieve this aim, we can only consider connected subgraphs p which are "closed" w.r.t. local and global constraints, i.e., such that $p \models c$ for all $c \in \mathcal{C}_L \cup \mathcal{C}_G$. We shall denote such graphs *weak patterns*, or simply w-patterns.

Example 3. Let us consider the workflow graph of Figure 1, and the following subgraphs.



p_1 p_2 p_3 p_4

p_1 and p_3 are not w-pattern: indeed, a is a deterministic fork (thus triggering the occurrence of node b), whereas l is an and-join (thus triggering the occurrence of both i and g). Notice that both p_2 and p_4 are instead w-patterns, since each constraint involving the contained nodes is satisfied. ◁

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 $\mathcal{WS}$.

Proposition 4 ([13]). *Let $p \subseteq \mathcal{CF}$. Then*

1. *deciding whether p is a pattern is* **NP***-complete.*
2. *given a multiset $\mathcal{F}$ of instances, deciding whether p is an $\mathcal{F}$-pattern or whether p is a w-pattern is computable in polynomial time in the size of $\mathcal{F}$.* □

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, since each frequent $\mathcal{F}$-pattern is bounded by w-patterns, as the following result states.

Lemma 1. Let p be a frequent $\mathcal{F}$-pattern. Then i) there exist a frequent w-pattern p' such $p \subseteq p'$, and ii) each weak pattern $p' \subseteq p$ is a frequent $\mathcal{F}$-pattern. □

We stress that a weak pattern is not necessarily an $\mathcal{F}$-pattern nor even a pattern. We shall use weak patterns to optimize the search space. The algorithm exploited uses a levelwise theory. Roughly speaking, we incrementally construct 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. The correctness follows from the results of Proposition 1, 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 deterministic closure of single nodes. A pattern is an elementary w-pattern (cf. *ew*-pattern) for a node a if it is the minimal (w.r.t. set inclusion) w-pattern containing a. 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. Moreover, given a set $E' \subseteq \mathrm{EW}$, $\mathit{Compl}(E') = \mathrm{EW} - \bigcup_{e \in E'} \mathrm{EW}_e$ contains all the elementary patterns which are neither in E' nor contained in some element of E'.

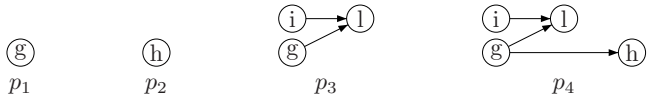
We now introduce a precedence relation $\prec$ among connected weak patterns. First of all, let us denote by $E^{\subseteq}$ the subset of arcs in $\mathcal{WS}$ whose source is not a deterministic fork, i.e., $E^{\subseteq} = \{(a,b) \in E \mid \mathtt{OUT}_{min}(a) < \mathit{OutDegree}(a)\}$. Given two connected w-patterns, say $p = \langle A_p, E_p \rangle$ and $p' = \langle A_{p'}, E_{p'} \rangle$, $p \prec p'$ if and only if:

a) $A_p = A_{p'}$ and $E_{p'} = E_p \cup \{(a,b)\}$, where $(a,b) \in E^{\subseteq} - E_p$ and $\mathtt{OUT}_{max}(a) > \mathit{OutDegree}_p(a)$ (i.e., p' can be obtained from p by adding an arc), or

b) there exists $p'' \in \mathit{Compl}(\mathrm{EW}_p)$ such that $p' = p \cup p'' \cup X$, where X is either empty if p and p'' are connected or contains exactly an edge in $E^{\subseteq}$ with endpoints in p and p'' (i.e., p' is obtained from p by adding an elementary weak pattern and possibly a connecting arc).

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

Example 4. With reference to the workflow graph of Figure 1, let us consider the subgraphs shown below:



The subgraphs p_1, p_2 and p_3 are elementary patterns: indeed, p_1 is the deterministic closure of g and p_2 is the deterministic closure of h, whereas p_3 can be obtained from l. Also, notice that $p_1 \subset p_3$. p_4 is not an elementary pattern, as no node can generate it. Notice that $p_2 \prec p_4$ and $p_3 \prec p_4$, since $p_4 = p_2 \cup p_3 \cup \{(g, h)\}$. $\lhd$

It can be shown that all the connected weak patterns of a given workflow schema can be constructed by means of a chain over the $\prec$ relation. As a consequence, 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 2, 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 $E^{\subseteq}$ (*addFrequentArc* function); or by adding an elementary weak patterns (*addEWFrequentPattern* function).

4 Mining Process Models

In this section we address the problem of inducing a model for a given process, based on data related to past executions. Let us assume that a workflow log $\mathcal{L}_P$ is given for a process P. In general, a process mining task consists in discovering a workflow schema $\mathcal{WS}(P)$, expressed through a suitable constrained graph, which describes the traces in $\mathcal{L}_P$. We are interested in devising a general approach which is independent of the particular syntax adopted for representing the global constraints. The solution we propose consists in discovering a set of alternative schemata having no global constraints, but collectively modelling the different behavioral patterns, instead of a single schema with global constraints explicitly expressed. To this purpose, we introduce the notion of *disjunctive workflow schema*.

A *disjunctive workflow schema* for a given process P, denoted by $\mathcal{WS}^{\vee}(P)$, is a set $\{\mathcal{WS}^1, ..., \mathcal{WS}^m\}$ of workflow schemata for P, with $\mathcal{WS}^j = \langle \mathcal{CF}^j, \mathcal{C}_L^j, \emptyset \rangle$, for $1 \leq j \leq m$. The *size* of $\mathcal{WS}^{\vee}(P)$, denoted by $|\mathcal{WS}^{\vee}(P)|$, is the number of

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};
2   k := 0, R := L_0;
3   FrequentArcs := {(a,b) | (a,b) ∈ E^⊆, ⟨{a,b},{(a,b)}⟩ is frequent w.r.t. F};
4   E^⊆_f := E^⊆ ∩ FrequentArcs;
5   repeat
6     U := ∅;
7     forall p ∈ L_k do begin
8       U := U ∪ addFrequentArc(p);
9       forall e ∈ Compl(EW_p) ∩ L_0 do
10        U := U ∪ addFrequentEWPattern(p, e);
11    end
12    L_{k+1} := {p | p ∈ U, p is frequent w.r.t. F};
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**;
$p' := \langle A_p \cup A_e, E_p \cup E_e\rangle$;
if p' is *connected*, **then return** p' **else return** *addFrequentConnection*(p', p, e);

Function *addFrequentConnection*$(p' = \langle A_{p'}, E_{p'}\rangle, p = \langle A_p, E_p\rangle, e = \langle A_e, E_e\rangle)$: **w-pattern**;
$S := \emptyset$
forall frequent $(a,b) \in E^{\subseteq}_f - E_p$ s.t. $(a \in A_p,\ b \in A_e) \vee (a \in A_e,\ b \in A_p)$ **do begin**
  $q := \langle A_{p'}, E_{p'} \cup (a,b)\rangle$;
  if $\mathcal{WS} \models q$ **then** $S := S \cup \{q\}$;
end
return S

Function *addFrequentArc*$(p = \langle A_p, E_p\rangle)$: **pattern**;
$S := \emptyset$
forall frequent $(a,b) \in E^{\subseteq}_f - E_p$ s.t. $a \in A_p$, $b \in A_p$ **do begin**
  $p' := \langle A_p, E_p \cup (a,b)\rangle$
  if $\mathcal{WS} \models p'$ **then** $S := S \cup \{p'\}$;
end
return S

Fig. 2. Algorithm w-find($\mathcal{F}$,$\mathcal{WS}$)

workflow schemata it contains. An instance of any $\mathcal{WS}^j$ is also an instance of $\mathcal{WS}^\vee$, denoted by $\mathcal{WS}^\vee \models I$. Moreover, a trace s which is compliant with any $\mathcal{WS}^j$ is also compliant with $\mathcal{WS}^\vee$, denoted by $\mathcal{WS}^\vee \models s$.

Hence, given $\mathcal{L}_P$, we aim at discovering a disjunctive schema $\mathcal{WS}^\vee$ as "close" as possible to the actual unknown schema $\mathcal{WS}(P)$ that generated the log, according to the following *soundness* and *completeness* notions. We define *soundness of* $\mathcal{WS}^\vee$ w.r.t. $\mathcal{L}_P$, the percentage of instances having corresponding traces in the log, i.e.,

$$soundness(\mathcal{WS}^\vee, \mathcal{L}_P) = \frac{|\{I \mid \mathcal{WS}^\vee \models I \wedge \exists s \in \mathcal{L}_P \text{ s.t. } \mathcal{WS}^\vee \models^I s\}|}{|\{I \mid \mathcal{WS}^\vee \models I\}|}$$

The *completeness of* $\mathcal{WS}^\vee$ w.r.t. $\mathcal{L}_P$, is instead the percentage of traces that are compliant with some trace in the log, ie.,

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

Thus, given two real numbers, namely α and σ, between 0 and 1, we say an induced schema $\mathcal{WS}^{\vee}$ is α-*sound* w.r.t. $\mathcal{L}_P$, if $soundness(\mathcal{WS}^{\vee}, \mathcal{L}_P) \leq \alpha$, whereas $\mathcal{WS}^{\vee}$ is σ-*complete* w.r.t. $\mathcal{L}_P$, if $completeness(\mathcal{WS}^{\vee}, \mathcal{L}_P) \geq \sigma$.

Notice that, for any value of α and σ, there always exists a trivial α-*sound* and σ-*complete* disjunctive schema $\mathcal{WS}^{\vee}$, consisting in the union of exactly one workflow (without global constraints) modelling each of the instances in $\mathcal{L}_P$. However, such model is not a syntectic view of the process P, for its size being $|\mathcal{WS}^{\vee}| = |\mathcal{L}_P|$. We thus introduce a bound on the size of $\mathcal{WS}^{\vee}$.

Then, given a workflow log $\mathcal{L}_P$ for the process P, a real number σ and a natural number m, we consider the following problem:

`MPD`(P, σ, m)**:** *Maximal Process Discovery*, i.e., find a σ-complete disjunctive workflow schema $\mathcal{WS}^{\vee}$, s.t. $|\mathcal{WS}^{\vee}| \leq m$ and $soundness(\mathcal{WS}^{\vee}, \mathcal{L}_P)$ is maximal.

Proposition 5 ([11]). `MPD`(P,σ,m) is an **NP**-complete optimization problem whose set of feasible solutions is not empty. □

Due to the above intractability result, the `MPD` problem is tackled with a greedy approach: in practice, we consider the variant `PD` problem, which consists in finding a σ-complete disjunctive schema with $|\mathcal{WS}^{\vee}| \leq m$, which is as sound as possible (i.e., a local optimum). In the rest of the section, we propose an efficient approach for solving the `PD` problem. The approach mainly relies on performing an iterative partitioning of the traces in the log, in order to find clusters of executions with a similar and unexpected (w.r.t. the local properties) behavior. Starting with a preliminary schema, which only accounts for the dependencies among the activities of P, the model is iteratively and incrementally refined by computing a specific workflow schema for each new cluster of traces. The schemata so obtained constitute a disjunctive workflow schema, which increases its soundness at each refinement step, still preserving its completeness. The algorithm exploits a "flat", relational representation of the traces obtained by projecting the instances on a suitable set of properly defined *features*.

The approach is encoded in the algorithm `ProcessDiscover`, shown in Figure 3, which computes a disjunctive schema $\mathcal{WS}^{\vee}$, taking as input a log $\mathcal{L}$ and three thresholds m, σ and $maxFeatures$ (which is an upper bound to the number of features that can be induced at each refinement step).

Notice that for the preliminary schema a control flow graph $\mathcal{CF}_{\sigma}$, expressing a minimal set of precedences with at least a given support σ, is computed through the procedure *minePrecedences* [1, 28]. Each workflow schema $\mathcal{WS}_i^j$, eventually inserted in $\mathcal{WS}^{\vee}$, is identified by the number i of refinements needed, and an index j distinguishing the schemata at the same refinement level. Moreover, we denote by $\mathcal{L}(\mathcal{WS}_i^j)$ the set of traces in the cluster defined by $\mathcal{WS}_i^j$. Notice that initially $\mathcal{WS}_0^1$, containing all the traces in $\mathcal{L}_P$, is put in $\mathcal{WS}^{\vee}$, and in Step 3 we refine the model by mining some local constraints, too.

At each step, function *refineWorkflow* is applied to a schema $\mathcal{WS}_i^j \in \mathcal{WS}^{\vee}$, chosen according a greedy heuristic: $\mathcal{WS}_i^j$ is the least sound schemata among the ones already discovered.

Input: A log $\mathcal{L}_P$, a real number σ, two natural numbers m and mF
Output: A disjunctive workflow schema $\mathcal{WS}^{\vee}$ (a solution of PD(P,σ,m)
Method: Perform the following steps:

1 $\mathcal{CF}_\sigma(\mathcal{WS}_0^1)$:=*minePrecedences*($\mathcal{L}_p$);
2 **let** $\mathcal{WS}_0^1$ be a schema, with $\mathcal{L}(\mathcal{WS}_0^1) = \mathcal{L}_P$;
3 *mineLocalConstraints*($\mathcal{WS}_0^1$);
3 $\mathcal{WS}^{\vee} := \mathcal{WS}_0^1$; *//Start clustering with the dependency graph only*
4 **while** $|\mathcal{WS}^{\vee}| < m$ **do**
5 $\mathcal{WS}_i^j$:=*leastSound*($\mathcal{WS}^{\vee}$);
6 $\mathcal{WS}^{\vee} := \mathcal{WS}^{\vee} - \{\mathcal{WS}_i^j\}$;
7 *refineWorkflow*(i,j);
8 **end while**
9 **return** $\mathcal{WS}^{\vee}$;

Procedure *refineWorkflow*(i: step, j: schema);

1 $\mathcal{F}$:=***identifyRelevantFeatures***($\mathcal{L}(\mathcal{WS}_i^j), \sigma, mF, \mathcal{CF}_\sigma$);
2 $\mathcal{R}(\mathcal{WS}_i^j)$:=***project***($\mathcal{L}(\mathcal{WS}_i^j), \mathcal{F}$);
3 $k := |\mathcal{F}|$;
4 **if** $k > 1$ **then**
5 $j := \max\{j \mid \mathcal{WS}_{i+1}^j \in \mathcal{WS}^{\vee}\}$;
6 $\langle \mathcal{WS}_{i+1}^{j+1}, ..., \mathcal{WS}_{i+1}^{j+k} \rangle := k\text{-}means(\mathcal{R}(\mathcal{WS}_i^j))$;
7 **for each** $\mathcal{WS}_{i+1}^h$ **do**
8 $\mathcal{WS}^{\vee} = \mathcal{WS}^{\vee} \cup \{\mathcal{WS}_{i+1}^h\}$;
9 $\mathcal{CF}_\sigma(\mathcal{WS}_{i+1}^h)$:=*minePrecedences*($\mathcal{L}(\mathcal{WS}_{i+1}^h)$);
10 *mineLocalConstraints*($\mathcal{WS}_{i+1}^h$);
11 **end for**
12 **else** *//Leaf of the tree*
13 $\mathcal{WS}^{\vee} = \mathcal{WS}^{\vee} \cup \{\mathcal{WS}_i^j\}$;
14 **end if**;

Function *identifyRelevantFeatures*(L: log, σ: threshold, mF: max nr. of features, $\mathcal{CF}_\sigma$: control flow graph): a set of minimal discriminant rules

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

Function *mostDiscriminantFeatures*($\mathcal{F}$: set of discriminant rules, mF: max nr. of features): set of discriminant rules;

1 $S' := \mathcal{L}$; $\mathcal{F}' := \emptyset$;
2 **do**
3 **let** $\phi = \texttt{argmax}_{\phi' \in \mathcal{F}} |w(\phi', S')|$;
4 $\mathcal{F}' := \mathcal{F}' \cup \{\phi\}$;
5 $S' := S' - w(\phi, S')$;
6 **while** $(|S'|/|\mathcal{L}_P| > \sigma)$ and $(\mathcal{F}' < mF)$;
7 **return** $\mathcal{F}'$;

Fig. 3. Algorithm `ProcessDiscover`

The function splits the traces of $\mathcal{WS}_i^j$ into k clusters, which are assigned to k distinct new schemata, $\mathcal{WS}_{i+1}^{j+1}, ..., \mathcal{WS}_{i+1}^{j+k}$ (where j is the maximum index of the schemata in $\mathcal{WS}^{\vee}$ with level $i+1$), which are put in $\mathcal{WS}^{\vee}$. For each schema

a control flow graph and a set of local constraints are derived, which suitably model the associated traces.

The algorithm `ProcessDiscover` converges in at most m steps, and exhibits the following interesting property.

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

The clustering of the log traces strongly relies on the procedures *identifyRelevantFeatures* and *project*. The former finds a set $\mathcal{F}$ of relevant features [21, 20, 22], whereas the latter projects the traces into a vectorial space whose components are, in fact, the mined features.

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

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

Example 5. In process *OrderManagament*, $[fil] \not\text{-}\!\!\rightarrow_{.3} m$ is a minimal discriminant rule, prescribing that fidelity discounts are never applied for new clients. Notice that $[dgl] \not\text{-}\!\!\rightarrow_{.3} o$ is a minimal discriminant rule as well. ◁

Again, the identification of the set $\mathcal{F}$ of discriminant rules can be carried out by a level-wise algorithm, as described in Figure 3.

The algorithm selects an optimal subset of features, with cardinality less or equal to $maxFeatures$, by exploiting the *mostDiscriminantFeatures* function, which works as follows. Let ϕ be a discriminant rule of the form $[a_i, ..., a_j] \not\text{-}\!\!\rightarrow_\sigma b$, then *the witness of ϕ in $\mathcal{L}$*, denoted by $w(\phi, \mathcal{L})$, is the set of logs in which the pattern $[a_i, ..., a_j]$ occurs. Then, the set of the most discriminant feature is computed through the heuristics of greedily selecting a feature ϕ covering the maximum number of traces, among the ones (S') not covered by previous selections.

5 Conclusions

In this paper we have introduced the problem of mining constrained graphs, with particular reference to the case of workflow systems. From an application viewpoint, the analysis of such models of execution can help in providing facilities for the human system administrator to monitor the actual behavior of many process models.

The paper proposes two distinct mining problems, and an overview of suitable solutions for such problems. In the context of inductive databases, the proposed problems raise interesting challenges, since the pattern languages introduced are worth even more complex mining tasks in which sophisticated constraints on the mining results can be specified. For example, one could be interested which discriminant factors characterize the failure or the success in the executions, or which is the choice that more frequently had led to a desired final configuration (e.g., to the acceptance of the order).

Interestingly, the techniques discussed in the previous sections are the adaptation of traditional learning techniques to a more structured domain in which background knowledge is available, and can be exploited for a smarter exploration of the search space. Indeed, frequent pattern discovery is essentially the adaptation of the *apriori* algorithm [2] to the case of workflow systems. Moreover, the Process Mining problem can be seen as a special case of inductive logic programming, in which the task is the mining of a set of consistent and complete clauses modelling the positive cases, and the latter correspond to log traces. Both the approaches presented in this paper have been extensively studied from an experimental point of view in [13, 12], thus demonstrating their effectiveness w.r.t. traditional approaches which do not properly exploit the available domain knowledge.

In this context, a challenging research direction is to extend the proposed techniques in a full multirelational setting. Indeed, the proposed model is essentially a *propositional* model, for it assumes a simplification of the constrained graphs 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'98)*, pages 469–483, 1998.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conference on Very Large Databases*, pages 487–499, 1994.
3. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 11th Int. Conf. on Data Engineering (ICDE95)*, pages 3–14, 1995.
4. D. J. Cook and L. B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research*, 1(1):231–255, 1994.
5. J.E. Cook and A.L. Wolf. Automating process discovery through event-data analysis. In *Proc. 17th Int. Conf. on Software Engineering (ICSE'95)*, pages 73–82, 1995.
6. J.E. Cook and A.L. Wolf. Event-based detection of concurrency. In *Proc. 6th Int. Symposium on the Foundations of Software Engineering (FSE'98)*, pages 35–45, 1998.
7. J.E. Cook and A.L. Wolf. Software process validation: quantitatively measuring the correspondence of a process to a model. *ACM Trans. Softw. Eng. Methodol.*, 8(2):147–176, 1999.

8. 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, University of Technology, Eindhoven, 2004. BETA Working Paper Series, WP 113.
9. L. Dehaspe and H. Toivonen. Discovery of Frequent DATALOG Patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
10. 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):119–153, 1995.
11. G.Greco, A.Guzzo, G.Manco, and D. Saccà. Mining frequent instances on workflows. In *Proc. 7th Pacific-Asia Conference (PAKDD'03)*, pages 209–221, 2003.
12. G.Greco, A.Guzzo, L.Pontieri, and D. Saccà. Mining expressive process models by clustering workflow traces. In *Proc. 8th Pacific-Asia Conference (PAKDD'04)*, pages 52–62, 2004.
13. G. Greco, A. Guzzo, G. Manco, and D. Saccà. Mining and reasoning on workflows. *IEEE Trans. on Data and Knowledge Eng.*, 17(4):519–534, 2005.
14. J. Han, J. Pei, and Y. Yi. Mining frequent patterns without candidate generation. In *Proc. Int. ACM Conf. on Management of Data (SIGMOD'00)*, pages 1–12, 2000.
15. J. Herbst. Dealing with concurrency in work?ow induction. In *Procs. European Concurrent Engineering Conference*, 2000.
16. J. Herbst and D. Karagiannis. Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. *Journal of Intelligent Systems in Accounting, Finance and Management*, 9:67–92, 2000.
17. A. Inokuchi, T. Washi, 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.
18. P. Koksal, S.N. Arpinar, and A. Dogac. Workflow history management. *SIGMOD Recod*, 27(1):67–75, 1998.
19. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. IEEE Int. Conf. on Data Mining (ICDM'01)*, pages 313–320, 2001.
20. H. Motoda and H. Liu. Data reduction: feature selection. *Handbook of data mining and knowledge discovery*, pages 208–213, 2002.
21. N.Lesh, M.J. Zaki, and M.Ogihara. Mining features for sequence classification. In *Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'00)*, pages 342–346, 1999.
22. B. Padmanabhan and A. Tuzhilin. Small is beautiful: discovering the minimal set of unexpected patterns. In *Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'00)*, pages 54–63, 2000.
23. R. Parekh and V. Honavar. Grammar Inference, Automata Induction and Language Acquisition. In *Handbook of Natural Language Processing*. Marcel Dekker, 2000.
24. J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. H-Mine: Hyper-structure mining of frequent patterns in large databases. In *Proc. IEEE Int. Conf. on Data Mining (ICDM'01)*, pages 441–448, 2001.
25. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE'2001)*, pages 215–224, 2001.
26. Guido Schimm. Mining most specific workflow models from event-based data. *Business Process Management*, pages 25–40, 2003.

27. W.M.P. van der Aalst and B.F. van Dongen. Discovering workflow performance models from timed logs. In *Proc. Int. Conf. on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, pages 45–63, 2002.
28. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G.Schimm, and A.J.M.M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
29. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.
30. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*. To appear.
31. X. Yan and J. Han. gSpan: Graph-based substructure pattern pining. In *Proc. IEEE Int. Conf. on Data Mining (ICDM'02)*, 2001. An extended version appeared as UIUC-CS Tech. Report: R-2002-2296.
32. X. Yan and J. Han. CloseGraph: Mining closed frequent graph patterns. In *Proc. ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, pages 286–295, 2003.
33. K. Yoshida, H. Motoda, and N. Indurkhya. Graph- based induction as a unified learning framework. *Journal of Applied Intel.*, 4:297–328, 1994.