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

Mining Unconnected Patterns in Workflows

G. Greco, A. Guzzo, G. Manco, D. Saccà

RT-ICAR-CS-04-05 **Aprile 2004**

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

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

Mining Unconnected Patterns in Workflows

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

Rapporto Tecnico N. 05:
RT-ICAR-CS-04-05

Data:
Aprile 2004

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

I rapporti tecnici dell'ICAR-CNR sono pubblicati dall'Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche. Tali rapporti, approntati sotto l'esclusiva responsabilità scientifica degli autori, descrivono attività di ricerca del personale e dei collaboratori dell'ICAR, in alcuni casi in un formato preliminare prima della pubblicazione definitiva in altra sede.

Mining Unconnected Patterns in 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. Recently data mining techniques have been proposed to predict the "most probable" workflow execution in order to discover which sequence of activities may lead to a successful execution and which others should be avoided for they often preluding an unsuccessful termination. Some previous work has addressed the problem with the aim of discovering frequent connected patterns in workflow. On the other side, the problem of discovering frequent unconnected patterns has not been dealt with despite of its relevance to the process of workflow mining: indeed finding sequences of activities which frequently occur together although they are not contiguous is crucial to discover meaningful execution patterns. This paper investigates the problem of mining unconnected patterns in workflows and presents for its solution two algorithms, both adapting the Apriori approach to the graphical structure of workflows. The first one is a straightforward extension of the level-wise style of Apriori whereas the second one introduces sophisticated graphical analysis of the frequencies of workflow instances. The experiments show that graphical analysis improves the performance of pattern mining by dramatically pruning the search space of candidate patterns.

1 Introduction

Workflow management systems (WfMs) represent the most effective technological infrastructure for managing business processes in several application domains [6, 13, 5]. There is a growing body of proposals aiming at enhancing this technology in order to provide facilities for the human system administrator while designing complex processes as well as in order to offer an "intelligent" support in the decisions which have to be done by the enteprise during the enactment [7, 3, 9, 12].

Within this line of research data mining techniques have proved to be very effective [15]. Traditionally, they have been employed for using the information collected during the enactment of a process not yet supported by a WfMS, such as the transaction logs of ERP systems like SAP, in order to derive a model explaining the events recorded [4, 14, 1]. Then, the output of these techniques, i.e., the "mined" synthetic model, can be profitably used to (re)design a detailed workflow schema, capable of supporting automatic enactments of the process. Under this perspective, these techniques offer a support in the design time of the workflow system and are called *process mining* techniques.

A novel line of research has been, instead, introduced in [8], by investigating the ability of predicting the "most probable" workflow execution. Indeed, in real world-cases, workflows are intrinsically non-deterministic, since they offer alternative executions which may lead to different results. Now, as actors make choices which influence the execution of a workflow, some choices may be beneficial, whereas others should be avoided in the future. In this perspective, data mining techniques can 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.

The algorithm presented in [8] is able to discover the connected structure of the executions, that have been scheduled more frequently by the workflow system, i.e., whose frequency of occurrence in the logs $\mathcal{F}$ is above a given threshold σ. These structures are simply called *frequent connected patterns* (short: frequent $\mathcal{F}$-patterns) and they are the subprocesses that are frequently performed during the enactment of the main process. The knowledge of the frequent $\mathcal{F}$-patterns can be profitably exploited by the administrator in order to identify anomalies in the enactment.

In this paper, we continue on this line of research by studying the problem of discovering some correlations among the mined subprocesses. Thus, we assume that a set $\mathcal{P}$ of frequent $\mathcal{F}$-patterns is given and we are interested in discovering whether any of the subsets of $\mathcal{P}$ is frequent as well. This problem, called *frequent unconnected patterns* discovery (short: FUPD), occurs very often in practical scenarios and is crucial for the identification of the critical subprocesses that led often to (un)desired final configuration. We show how the structure of the workflow together with some elementary information such as the frequency of occurrences of elementary activities suffices for pruning the search space and for deriving an efficient and practically fast algorithm, called *ws^*-unconnected-find*.
Organization. The rest of the paper is organized as follows. In the next section, we define the formal model of workflow and we introduce the problem of mining frequent unconnected patterns. In Section 3 we introduce an a-priori like algorithm, while the *ws^*-unconnected-find* algorithm is shown in Section 4. The description of the implementation of both approaches is reported, while Section 5 discusses of several experiments that confirm the validity of the approach. Finally in Section 6 we draw our conclusions by pointing to further enhancements to the proposed approach that are worth future research efforts.

2 Workflow Model and Problem Formulation

A *workflow schema* $\mathcal{WS}$ is a tuple $\langle A, E, a_0, A_F, A^{\vee}_{in}, A^{\wedge}_{in}, A^{\vee}_{out}, A^{\wedge}_{out}, A^{\otimes}_{out}\rangle$, where A is a finite set of *activities*, $E \subseteq (A - A_F) \times (A - \{a_0\})$ is an acyclic relation of precedences among activities, $a_0 \in A$ is the starting activity, $A_F \subseteq A$ is the set of final activities, $A^{\vee}_{in}$ are the or-join nodes in A, $A^{\wedge}_{in}$ are the and-join nodes in A, $A^{\vee}_{out}$ are the or-fork nodes in A, $A^{\wedge}_{out}$ are the and-fork nodes in A, and $A^{\otimes}_{out}$ are the exclusive-fork nodes in A. The tuple $\langle A, E\rangle$ is often referred to as the *control graph* of $\mathcal{WS}$.

Informally, an activity in $A^{\wedge}_{in}$ 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 $A^{\vee}_{in}$ can start as soon as at least one of its predecessors has been completed. Moreover, once finished, an activity a in $A^{\wedge}_{out}$ activates all its outgoing activities, a in $A^{\vee}_{out}$ activates some of the outgoing activities, while a in $A^{\otimes}_{out}$ activates exactly one outgoing activity.

Example 1. An example of workflow schema is shown in Figure 1. In this schema, we adopt the graphical convention of representing nodes in $A^{\vee}_{in}$ with circles and nodes in $A^{\wedge}_{in}$ with bold circles; moreover, nodes in $A^{\otimes}_{out}$ exhibit dashed outgoing arcs, whereas nodes in $A^{\vee}_{out}$ exhibit dotted arcs and nodes in $A^{\vee}_{out}$ exhibit bold arcs. Finally, nodes in A_F are represented by means of a double circle. To summarize figure 1 represents a schema $\mathcal{WS}$ where $A^{\wedge}_{in} = \{d, c, b, f, e, n\}$ and $A^{\vee}_{in} = \{a, g, l, i, h, m, o, p\}$, while $A^{\otimes}_{out} = \{a, h\}$, $A^{\wedge}_{out} = \{l, i, g, e, f, m, n, o\}$, $A^{\vee}_{out} = \{b, c, d\}$, and $A_F = \{p\}$. $\triangleleft$

The enactment of a workflow gives rise to an instance, i.e., to a proper subgraph of the schema which is derived satisfying the constraints imposed by the instances included.

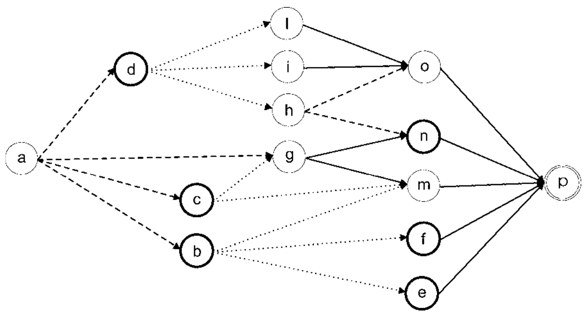


Fig. 1. An example of workflow schema.

Definition 1 (Instance). Let $\mathcal{WS}$ be a workflow schema. Any connected subgraph $I = \langle A_I, E_I \rangle$ of the control flow graph, such that **(i)** $a_0 \in A_I$, **(ii)** $A_I \cap A_F \neq \emptyset$, **(iii)** for each $a \in A_I$, $|\{b \mid (b,a) \in E_I\}| > 0$, **(iv)** for each $a \in A_I \cap A^{\wedge}_{in}$, $\{b \mid (b,a) \in E\} \subseteq A_I$, **(v)** for each $a \in A_I \cap A^{\wedge}_{out}$, $\{b \in A^{\vee}_{in} \mid (a,b) \in E\} \subseteq A_I$, and **(vi)** for each $a \in A_I \cap A^{\otimes}_{out}$, $|\{b \mid (a,b) \in E_I\}| \leq 1$ and $|\{b \mid (a,b) \in E_I\}| = 1$ if $\{b \in A^{\vee}_{in} \mid (a,b) \in E\} \neq \emptyset$, is an *instance* of $\mathcal{WS}$ ($\mathcal{WS} \models I$). □

Example 2. With reference to the schema of fig. 1, the following are example instances:

$$\begin{tikzcd}
 & & f \arrow[dr] & \\
a \arrow[r] & b \arrow[ur] \arrow[dr] & & p \\
 & & m \arrow[ur] &
\end{tikzcd}
\quad
\begin{tikzcd}
 & & n \arrow[dr] & \\
a \arrow[r] & g \arrow[ur] \arrow[dr] & & p \\
 & & m \arrow[ur] &
\end{tikzcd}
\quad
\begin{tikzcd}
 & & & n \arrow[dr] & \\
a \arrow[r] & c \arrow[r] & g \arrow[ur] \arrow[dr] & & p \\
 & & & m \arrow[ur] &
\end{tikzcd}$$

$a \to d \to l \to o \to p$ $\quad$ $a \to d \to i \to o \to p$ $\quad$ $a \to d \to h \to o \to p$

$a \to b \to e \to p$ $\quad$ $a \to b \to f \to p$ $\quad$ $a \to b \to e \to p$ $\quad$ $a \to c \to m \to p$ ◁

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 $\mathcal{WS}$.

Among the instances of $\mathcal{F}$ we are interested in discovering the most frequent patterns of execution as next defined.

Definition 2 (Pattern). A graph $p = \langle A_p, E_p \rangle \subseteq \mathcal{WS}$ 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*, we consider the following two relevant problems on workflows:

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

FUPD: *Frequent Unconnected Pattern Discovery*, i.e., finding all the subsets of connected patterns whose support is greater than *minSupp*.

We remark that FCDP has been addressed in [8]. Then, in this paper we shall deal with an efficient solution for FUDP by assuming that the set $C(\mathcal{F})$ of all the frequent (w.r.t. *minSupp*) connected patterns in the set of instances $\mathcal{F}$ has been already computed using the approach of [8]. It is worth noting that FUPD has a straightforward solution consisting in the application of a level-wise algorithm (in the *a-priori* style) [2, 11] which combines all the unconnected patterns in $\mathcal{P}$

and then checks for their frequency. Indeed in Section 3 we present an implementation of Apriori that takes into account some basic properties of workflows. However, in order to achieve a larger amount of pruning of the search space, we need to further exploit the peculiarities of the workflow graph. To this end, one might think of combining the dynamic information obtained from the frequency of single nodes and edges, with the static information derived from the workflow schema in order to predict (un)frequent patterns. To get an intuition of the approach, consider again the schema in Figure 1. Observe that the activities a and p are frequent but not necessarily any path from a to p is frequent as well (this is what happens, e.g., by considering the instances of example 2 and $minSupp = 30\%$). On the other hand, as every execution starting from a will eventually terminate in p, we can then conclude that the frequency of any pattern containing a remains equal if the pattern is extended with p. Therefore, we can conclude that nodes a and p form a frequent unconnected pattern without looking at the actual co-occurrences in the log files.

Actually, many situations are less evident than the above trivial case. For instance, by analyzing both the instances and the graph structure (with the techniques we shall develop in the paper), we are also able to conclude that m frequently occurs together with a. Incidentally, note also that m and b cannot co-occur frequently, since the only path connecting them is below the frequency threshold (and hence the frequency of m cannot be related to that of b). In order to systematically study such circumstances, we develop a graph theoretic approach for predicting whether two activities are coupled just on the basis of the workflow structure and of the frequency of the elementary activities alone.

3 A Level-Wise Algorithm for Unconnected Patterns

In this section we present a first simple solution to the FUPD problem, achieved by means of the algorithm *ws-unconnected-find* shown in Figure 2. The algorithm receives in input the workflow schema $\mathcal{WS}$ and the set $C(\mathcal{F})$ of frequent connected $\mathcal{F}$-patterns, which are assumed to be already computed, and returns all the frequent unconnected patterns.

Before detailing the mains steps we need some further definitions and notations. Given a unconnected pattern p, we say that p is a *starting* pattern if it contains the starting activity of the workflow schema; otherwise, it is said *terminating* pattern. Rather than computing all the possible unconnected patterns, we limit on starting patterns and we show how the space of all the connected starting patterns forms a lower semi-lattice that can be profitably explored in a bottom-up fashion. In fact, given two starting patterns r and p we say that r directly precedes p, denoted by $r \prec p$, if there exist a terminating pattern q such that $r = p \cup q$. Moreover, r precedes p, denoted by $r \prec^* p$, if either $r \prec p$ or there exists a starting pattern q such that $r \prec^* q$ and $q \prec^* p$. It is not difficult to see that starting patterns can be constructed by means of a chain over the $\prec$ relation. Such an approach is, in fact, exploited by the algorithm in Figure 2 that computes all the frequent starting patterns, by generating at each step k the patterns made of k distinct unconnected patterns (stored in the set L_k).

The algorithm starts by defining L_0 as the set of frequent patterns in $C(\mathcal{F})$ that contains a_0, and C' to be the set of all the terminating connected patterns — notice that the set is, in fact, $C(\mathcal{F})$ minus the starting patterns in L_0.

Then, at each step it generates a number of candidates (stored in U) in the main cycle (steps 4–13). Each generated pattern p is obtained by the function `UpdateCandidateList`, by combining a starting pattern in L_k with a connected terminating pattern q in C' which is not in the set $discarded(p)$. This latter set is used for optimization purposes. In fact, since we are interested in unconnected components, given a pattern p we can compute in advance a set of connected patterns that must not be combined with p, denoted by $discarded(p)$. This set contains all the patterns which have a non-null intersection with p, and it is initialized in the procedure `InitializeStructures`.

Input: A workflow schema $\mathcal{WS}$, a set $\mathcal{F}$ of instances of $\mathcal{WS}$, the minimal support *minSupp*, the set $C(\mathcal{F})$ of frequent connected $\mathcal{F}$-patterns.
Output: A set of frequent unconnected $\mathcal{F}$-patterns.
Method: Perform the following steps:

```
1   InitializeStructures();
2   $L_0 := \{ p \mid p \in C(\mathcal{F}),\ a_0 \in p \}$;        //***frequent connected starting patterns
3   $k := 0, R := L_0;\ C' := C(\mathcal{F}) - L_0$;
4   repeat
5     $U :=$ UpdateCandidateList($L_k$)
6     $L_{k+1} :=$ ComputeFrequentPatterns(U);
7     $R := R \cup L_{k+1}$;
8     forall $r \in U - L_{k+1}$ do begin
9       $p := starting(r)$;
10      forall $p' \in L_{k+1}$ s.t. $p \subset p'$ do
11        $discarded(p') := discarded(p') \cup \{terminating(r)\}$;
12    end
13  until $L_{k+1} = \emptyset$;
14  return $R$;
```

```
Procedure InitializeStructures;
  IS1  forall $p \in C(\mathcal{F})$ do
  IS2    $discarded(p) := \{ q \mid q \in C(\mathcal{F}),\ p \cap q \neq \emptyset \}$;
```

```
Function ComputeFrequentPatterns($U$: set of candidates): set of frequent patterns;
  CFP1  return $\{ r \mid r \in U,\ supp(r) > minSupp \}$;
```

```
Function UpdateCandidateList($L_k$: set of candidates): set of candidate patterns;
  UCL1  $U := \emptyset$
  UCL2  forall $p \in L_k$ do                              //***starting pattern
  UCL3    forall $q \in C' - discarded(p)$ do begin        //***terminating pattern
  UCL4      $r := p \cup q$; $starting(r) = p$; $terminating(r) = q$;
  UCL5      $discarded(r) := discarded(p) \cup discarded(q)$;
  UCL6      $U := U \cup \{r\}$;
  UCL7    end;
  UCL8  return $U$;
```

Fig. 2. Algorithm ws-*unconnected-find*($\mathcal{F}$,$\mathcal{WS}$,*minSupp*,$C(\mathcal{F})$)

Moreover, notice that each pattern r generated at the step k is also equipped with two functions, $starting(r)$ and $terminating(r)$, which store the starting and terminating patterns respectively that have been used for generating r.

After all the candidates have been computed in the set U, the function `ComputeFrequentPatterns` is invoked (step 6) for filtering the elements in U which frequently occur in $\mathcal{F}$, thus creating the set L_{k+1} containing all the frequent unconnected patterns made of $k + 1$ unconnected patterns — notice that in this implementation this task is simply done by means of a scan in the logs $\mathcal{F}$.

Finally, the generated starting frequent patterns are added to the actual result R, and in the steps 8–12 the set $discarded$ is updated for the patterns which are discovered to be frequent.

The computational cost of the algorithm is related to the number of unconnected components contained by the maximal frequent unconnected patterns. This number indeed influences the number of scans to the log file.

Proposition 1. *The algorithm* ws-unconnected-find *computes the set of all the unconnected frequent patterns with at most* $|C(\mathcal{F})| - |L_0|$ *scans in the log file.* □

The correctness of the algorithm follows from the following observation.

Theorem 1. *For any two patterns* p *and* q *such that* $p \cup q$ *is a frequent unconnected pattern, there exists a pattern* p' *containing both* p *and the initial activity* a_0 *such that* $p' \cup q$ *is frequent as well.* □

Informally, the theorem states that, since the starting activity is executed in each instance, each unconnected frequent pattern can be extended with the initial activity. As a consequence, each frequent unconnected pattern can be generated starting from the unconnected frequent starting patterns.

Example 3. By assuming $minSupp = 30\%$ and the set $\mathcal{F}$ of instances of example 2, the patterns $\{\mathtt{a},\mathtt{b}\} \cup \mathtt{p}$, $\mathtt{a} \cup \mathtt{m} \cup \mathtt{p}$ and $\{\mathtt{a},\mathtt{d}\} \cup \mathtt{o} \cup \mathtt{p}$, are maximal unconnected frequent patterns. □

4 Optimizing Candidate Generation

In this section we present some techniques for efficiently pruning the search space which has been identified by means of the level-wise algorithm *ws-unconnected-find*. Our idea is to exploit the structure and the information regarding the frequency of each activity in order to identify, before their actual testing w.r.t. the logs, those patterns which are necessarily (un)frequent.

In the following, we assume the existence of a set $\mathcal{F}$ of instances of a workflow schema $\mathcal{WS}$. Then, let q be a non-necessarily connected component of $\mathcal{WS}$ with frequency $f(q)$ and p be a connected component with frequency $f(p)$ such that q and p are unconnected. Our aim is to compute as efficiently as possible the number of instances in $\mathcal{F}$ executing both the components p and q, denoted by $f_p(q)$.

Obviously, the most trivial and inefficient way for computing $f_p(q)$ is to make a scan of the log $\mathcal{F}$. However, we shall show how some proper data structures and algorithms can be used for effectively identifying a suitable lower bound and an upper bound for $f_p(q)$, denoted by $l_p(q)$ and $u_p(q)$ respectively, in some efficient way not requiring the access to the log.

We next start with the basic situation in which p and q are patterns each one made of a single activity of $\mathcal{WS}$.

4.1 Computing Frequency Bounds for Activities

Given an activity $a \in A$, let G_a be the subgraph of the control flow of $\mathcal{WS}$ induced by all the nodes b such that there is a path from b to a in $\mathcal{WS}$. Note that all such nodes can be easily determined by reversing the arcs in $\mathcal{WS}$ and computing the transitive closure of a.

The starting point of our approach is to compute for each node b in G_a, the number of instances in $\mathcal{F} = \{I_1, ..., I_n\}$ executing both the activities a and b, denoted by $f_a(b)$. As already said, we actually turn for computing a lower bound $l_a(b)$ and an upper bound $u_a(b)$ — obviously $l_a(b) \leq u_a(b) \leq min(f(a), f(b))$.

In order to accomplish this task we need some auxiliary data structures besides the workflow schema which are used for storing the occurrences of each activity and edge (connecting activities) in the log $\mathcal{F}$.

Definition 3 (Frequency graph). Let $\langle A, E\rangle$ be the control flow of a workflow schema $\mathcal{WS}$ and let $\mathcal{F} = \{I_1, ..., I_n\}$ be a set of instances of $\mathcal{WS}$. The *frequency graph* $\mathcal{WS}_{\mathcal{F}} = \langle A, E, f_A, f_E\rangle$ is a weighted graph such that

- $f_A : A \mapsto \mathbb{N}$ maps each activity a to the number of instances in $\mathcal{F} = \{I_1, ..., I_n\}$ executing it, and
- $f_E : E \mapsto \mathbb{N}$ maps each arc e to the number of instances in $\mathcal{F} = \{I_1, ..., I_n\}$ containing this arc.

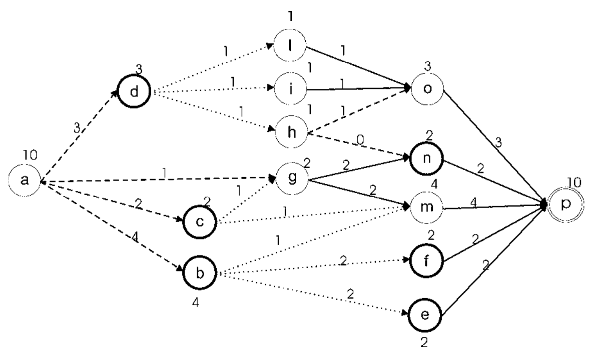


Fig. 3. Frequency graph associated with the schema of fig. 1.

Whenever no confusion arises, given an activity $a \in A$ (resp. an edge $e \in E$), $f_A(a)$ (resp. $f_E(e)$) will be simply denoted by $f(a)$ (resp. $f(e)$). □

Figure 3 shows the frequency graph associated with the schema of fig. 1, built by taking into account the set $\mathcal{F}$ of instances described in example 2.

In order to derive these bounds, we first determine a topological sort $\langle a = b_1, b_2, \ldots, b_k \rangle$ of the nodes in G_a of $\mathcal{WS}$ - as $\mathcal{WS}$ is acyclic a topological sorts exists for each of its subgraphs including G_a. Then we proceed as shown in Figure 4. In the step 1, the lower bound and the upper bound of the activity a is obviously fixed to the known value $f(a)$, determined through G_a. Then, each node b_i in G_a is processed according to the topological sort. In 3, the set of all the activities $C(b_i)$ that can be reached be means of an edge starting in b_i and that are in G_a is computed – note that $|C(b_i)| \neq 0$. Step 4 is responsible for computing the upper bound $u_a(b_i)$, whereas steps 5–9 are responsible for computing the lower bound $l_a(b_i)$. Intuitively, the upper bound $u_a(b_i)$ can be computed by optimistically assuming that each arc outgoing from b_i is in some path reaching c. This justifies the formula of step 4.

Concerning $l_a(b_i)$, observe that each node $c_j \in C(b_i)$ is executed with a by at least $l_a(c_j)$ instances. Therefore, we need to know how many of the instances executing b_i contribute to $l_a(c_j)$. Two cases arise: (i) $b_i \in A^{\vee}_{out} \cup A^{\wedge}_{out}$, so the nodes connected to b_i may occur simultaneously within an instance, and (ii) $b_i \in A^{\otimes}_{out}$, then all c_j are executed exclusively from each other. This explains why in the first alternative $L^{\wedge}_1$ and $L^{\vee}_1$ are computed by maximizing the contribution of each c_j, whereas in the second alternative the single contributions are summated. Finally, observe that when $c_j \in A^{\vee}_{in}$, it may be not the case that all of the $l_a(c_j)$ instances execute b_i, thus requiring to differentiate the formulas for $L^{\vee}_1$ and $L^{\wedge}_1$ (and, in the same way, for $L^{\vee}_2$ and $L^{\wedge}_2$).

Observe that the final step in the algorithm possibly find tighter lower bounds by exploiting the fact that, given two nodes b and c in G_a if $(b, c) \in \mathcal{WS}$, b is an and-fork node and c is an or-join node, then $l_a(c) \leq l_a(b)$ the activity b is executed each time the activity c is.

Theorem 2. *The following properties hold for the algorithm in Figure 4:*

1. *The parameters U, $L^{\vee}_1$, $L^{\vee}_2$, $L^{\wedge}_1$ and $L^{\wedge}_2$ are well defined, i.e., $u_a(b_i)$ and $l_a(b_i)$ are computed by exploiting already processed values.*
2. *For each node $b_i \in G_i$, the values $l_a(b_i)$ and $u_a(b_i)$ are, respectively, lower and upper bound of the frequency $f_a(b_i)$.*
3. *The procedure can be computed in time $O(|G_a|^2)$.* □

Example 4. By applying the above formulas, we obtain the following bounds for node m:

$$l_{\mathrm{m}}(\mathrm{a}) = 3,\quad l_{\mathrm{m}}(\mathrm{b}) = 1,\quad l_{\mathrm{m}}(\mathrm{c}) = 1,\quad l_{\mathrm{m}}(\mathrm{d}) = 0,\quad l_{\mathrm{m}}(\mathrm{g}) = 2,\quad l_{\mathrm{m}}(\mathrm{h}) = 0,$$
$$u_{\mathrm{m}}(\mathrm{a}) = 4,\quad u_{\mathrm{m}}(\mathrm{b}) = 1,\quad u_{\mathrm{m}}(\mathrm{c}) = 2,\quad u_{\mathrm{m}}(\mathrm{d}) = 0,\quad u_{\mathrm{m}}(\mathrm{g}) = 2,\quad u_{\mathrm{m}}(\mathrm{h}) = 0.$$

Input: A workflow schema $\mathcal{WS}$, an activity a, the graph G_a and a topological sort $\langle a = b_1, b_2, \ldots, b_k \rangle$ of the nodes in G_a.
Output: for each node $b \in G_a$, the values $l_a(b)$ and $u_a(b)$.
Method: Perform the following steps:

```
1   $l_a(a) := f(a);\ \ u_a(a) := f(a);$
2   **forall** $i = 2..k$ **do begin**
3     $C(b_i) := \{b \mid (b_i, b) \in E \wedge b \in G_a\};$
4     $u_a(b_i) := \min(f(b_i), f(a), U)$, with $U = \sum_{e=(b_i,c) \mid c \in C(b_i)} \min(f(e),\ u_a(c))$.
5     **if** $b_i \in A^{\vee}_{out} \cup A^{\wedge}_{out}$ **then**
6       $l_a(b_i) := \min(f(b_i), \max(L^{\wedge}_1, L^{\vee}_1))$, with
          $L^{\wedge}_1 = \max_{c_j \in C(b_i) \cap A^{\wedge}_{in}} \{l_a(c_j)\}$, and
          $L^{\vee}_1 = \max_{c_j \in C(b_i) \cap A^{\vee}_{in}} \{\max(0,\ l_a(c_j) - \sum_{e=(d,c_j) \in \mathcal{WS} \wedge\, d \neq b_i} f(e))\}$
7     **else** *// case of* $b_i \in A^{\otimes}_{out}$
8       $l_a(b_i) := \min(f(b_i), L^{\wedge}_2 + L^{\vee}_2)$, with
          $L^{\wedge}_2 = \sum_{c_j \in C(b_i) \cap A^{\wedge}_{in}} \{l_a(c_j)\}$, and
          $L^{\vee}_2 = \sum_{c_j \in C(b_i) \cap A^{\vee}_{in}} \{\max(0,\ l_a(c_j) - \sum_{e=(d,c_j) \in \mathcal{WS} \wedge\, d \neq b_i} f(e))\}$
9     **end**
10  **end**
11  **forall** $(b, c) \in G_a$ **do begin**
12    **if** $b \in A^{\wedge}_{out}$ and $c \in A^{\vee}_{in}$ **then**
13      $l_a(c) := \max(l_a(c), l_a(b))$
14  **endfor**
```

Fig. 4. Algorithm $compute\_frequency\_bounds(\mathcal{WS}_{\mathcal{F}}, a)$

According to the these bounds, it is easy to see that $\mathtt{m} \cup \mathtt{a}$ is a frequent unconnected pattern, whereas $\mathtt{m} \cup \mathtt{b}$ and $\mathtt{m} \cup \mathtt{d}$ are not (even though $\mathtt{b}$, $\mathtt{d}$ and $\mathtt{m}$ are frequent patterns). It is interesting to analyze also the bounds for node $\mathtt{o}$:

$$\begin{array}{lllll} l_{\mathtt{o}}(\mathtt{a}) = 1, & l_{\mathtt{o}}(\mathtt{d}) = 1, & l_{\mathtt{o}}(\mathtt{h}) = 1, & l_{\mathtt{o}}(\mathtt{i}) = 1, & l_{\mathtt{o}}(\mathtt{l}) = 1, \\ u_{\mathtt{o}}(\mathtt{a}) = 3, & u_{\mathtt{o}}(\mathtt{d}) = 3, & u_{\mathtt{o}}(\mathtt{h}) = 1, & u_{\mathtt{o}}(\mathtt{i}) = 1, & u_{\mathtt{o}}(\mathtt{l}) = 1 \end{array}$$

Thus, even if $\mathtt{d} \cup \mathtt{o}$ is a frequent unconnected pattern, lower bounds do not help in detecting such pattern without resorting to the logs. This is essentially due to the fact that, since $\mathtt{d} \in A^{\vee}_{out}$, its lower bound depends from the lower bounds of $\mathtt{h}$, $\mathtt{i}$ and $\mathtt{l}$ (each of which belongs to $G_{\mathtt{o}}$, with frequency 1). $\triangleleft$

4.2 Computing Frequency Bounds for Patterns

Let us now turn to the more general problem of approximating the value of $f_p(b)$, for any pattern p and any activity b, by means of a suitable lower and upper bound. Notice that the value $f_p(b)$ is the number of instances in $\mathcal{F}$ executing both the component p and each activity b that precedes one of the activities in p.

To this aim we simply reuse the technique described in the previous section with some adaptations. Let $INBORDER(p)$ denote the set of the activities in p which have no incoming arcs in p. Let $\mathcal{WS}(p)$ be the workflow schema derived from $\mathcal{WS}$ by adding a new and-join node, say a_p, corresponding to the component p, and by adding an arc from each node b in $INBORDER(p)$ to a. In the frequency graph of $\mathcal{WS}(p)$ set $f(a_p) = f(p)$, and $f(e) = f(p)$ for each $e = (b, a_p) \in E$. Then, the function $compute\_frequency\_bounds(\mathcal{WS}_{\mathcal{F}}, p)$ is defined as $compute\_frequency\_bounds(\mathcal{WS}(p)_{\mathcal{F}}, a_p)$.

Theorem 3. *Let $\mathcal{WS}$ be a workflow schema, $\mathcal{F}$ be a set of instances, and p a pattern. For any activity b, let $l_{a_p}(b)$ and $u_{a_p}(b)$ be the lower and upper bound of the occurrence of activity a_p together with b, computed by means of the algorithm $compute\_frequency\_bounds(\mathcal{WS}(p)_{\mathcal{F}})$. Then, $l_{a_p}(b)$ and $u_{a_p}(b)$ are indeed lower and upper bounds of $f_p(b)$.* □

4.3 Algorithm *ws*-unconnected-find*

Once the frequency bounds for a given pattern (w.r.t. any activity) are computed, we can face the more general problem. Let q be a general component of $\mathcal{WS}$ with frequency $f(q)$ and p be a connected component with frequency $f(p)$ such that q and p are unconnected. A lower bound and an upper bound of $f_p(q)$ are as follows:

- $l_p(q) = \max(0, \max_{b \in q}\{l_p(b) - (f(b) - f(q))\})$
- $u_p(q) = \min(f(q), \sum_{b \in \texttt{OUTBORDER}(q)} u_p(b))$.

Here, $\texttt{OUTBORDER}(p)$ refers to all the nodes in q having outgoing arcs in $\mathcal{WS} - q$. The intuition behind the above formulas is the following. The value $u_p(q)$ is obtained by taking into account the contribution of each node b of q from which there is a path to a node in p. However we may exclude in the upper bound computation all internal nodes of q (i.e., those not in $\texttt{OUTBORDER}(p)$) as they are always executed together with at least one node in $\texttt{OUTBORDER}(p)$. Concerning the computation of $l_p(q)$, observe that there are at least $l_p(b)$ instances executing $b \in q$ and p. So, as $f(b) \geq f(q)$, there are at least $l_p(b) - (f(b) - f(q))$ instances connecting q and p and executing b. It turns out that a suitable lower bound is provided by the node exhibiting the maximum such value.

Theorem 4. *Let $\mathcal{WS}$ be a workflow schema, $\mathcal{F}$ be a set of instances, and p and q two patterns. Then, $l_p(q)$ and $u_p(q)$ are lower and upper bounds of $f_p(q)$.* □

Generalized upper and lower bounds can be finally used for pruning the search space of the *ws-unconnected-find* algorithm. In fact, if for any two patterns $u_p(q) <$ *minSupp* then it is always the case that p and q never occur frequently together. Conversely, if $l_p(q) \geq$ *minSupp* then p and q can be combined into a pattern that is frequent as well.

Thus, the algorithm *ws-disconected-find* can be optimized (see Figure 5), by suitably adapting the procedures `InitializeStructures`, `UpdateCandidateList` and `ComputeFrequentPatterns`. Specifically, the former also compute the frequency graph and all the frequency bounds for any pattern, by exploiting the above formulas. The second, instead, verifies the frequency in the log $\mathcal{F}$ only for patterns which cannot be tested with the frequency bounds only.

5 Experiments and Discussion

In this section we study the behavior of the *ws*-unconnected-find* algorithm, by examining its pruning capability in experiments aimed at evaluating whether the computation of upper and lower bounds avoids the generation of unnecessary candidate patterns to check for frequency against the log data.

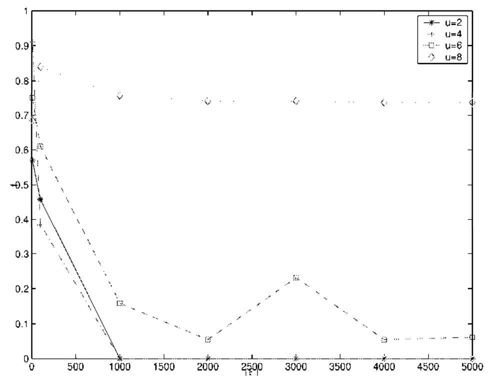
In our experiments, we use synthesized data whose generation can be tuned according to: i) the size of $\mathcal{WS}$, ii) the size of $\mathcal{F}$, iii) the average number d of connected frequent patterns to use in the generation of frequent unconnected patterns, and iv) the average number u of frequent unconnected patterns to exploit in the generation of unfrequent unconnected patterns. Data are generated according to the following strategy. First, a number d of frequent connected patterns are generated; next, iteratively, a pair p, q of frequent patterns is randomly chosen and merged

```
Procedure InitializeStructures;
IS1  $\mathcal{WS}_{\mathcal{F}} := compute\_frequency\_graph(\mathcal{WS}, \mathcal{F})$;
IS2  forall $p \in C(\mathcal{F})$ do begin
IS3    $discarded(p) := \{\, q \mid q \in C(\mathcal{F}),\ p \cap q \neq \emptyset \,\}$;
IS4    $\langle l_p, u_p \rangle := compute\_frequency\_bounds(\mathcal{WS}_{\mathcal{F}}, p)$
IS5  end;
```

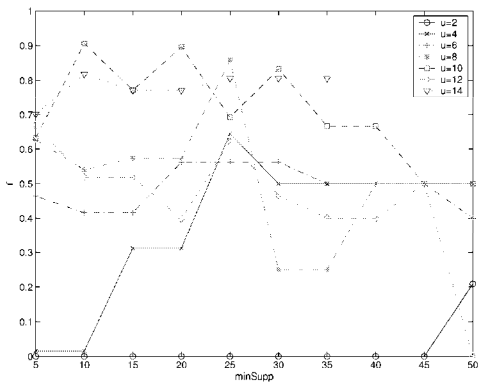
```
Function ComputeFrequentPatterns($U$: set of candidates): set of frequent patterns;
CFP1  $LF := \{\, r \mid r \in U,\ l_{terminating(r)}(starting(r)) \geq minSupp \}$;
CFP2  $LU := \{\, r \mid r \in U,\ u_{terminating(r)}(starting(r)) < minSupp \}$;
CFP3  return $LF \cup \{\, r \mid r \in U - (LF \cup LU),\ r$ is frequent w.r.t. $\mathcal{F} \,\}$;
```

```
Procedure UpdateCandidateList($U$: set of candidates);
UCL1  forall $p \in L_k$ do                              //***starting pattern
UCL2    forall $q \in C' - discarded(p)$ do begin        //***terminating pattern
UCL3      $r := p \cup q$; $starting(r) = p$; $terminating(r) = q$;
UCL4      $discarded(r) := discarded(p) \cup discarded(q)$;
UCL5      $l_q(p) = \max(0, \max_{b \in p}\{l_q(b) - (f(b) - f(p))\})$;
UCL6      $u_q(p) = \min(f(p), \sum_{b \in \mathtt{OUTBORDER}(q)} u_p(b))$;
UCL7      $U := U \cup \{r\}$;
UCL8    end;
```

Fig. 5. Algorithm *ws^*-unconnected-find($\mathcal{F}$,$\mathcal{WS}$,minSupp,$C(\mathcal{F})$)*



(a) Ratio f, increasing values of $|\mathcal{F}|$ and u.



(b) Ratio f, increasing values of $minSupp$ and u.

Fig. 6. Performance Graphs.

into $r = p \cup q$ by connecting OUTBORDER(p) to INBORDER(q) in such a way that $p \cup q$ is frequent but unconnected. More in details, let f_p and f_q be the frequencies of p and q, respectively. Each node in OUTBORDER(p) is connected to a new node $a \in A^{\vee}_{in} \cap A^{\otimes}_{out}$. Similarly, a new node $b \in A^{\vee}_{in} \cap A^{\otimes}_{out}$ is connected to each node in INBORDER(q). f_r is then set to $\max(f_p, f_q)$, and a connection between a and b is set by adding at most $n = \min(f_p, f_q)$ unfrequent nodes to r, and by connecting a and b by means of paths traversing such nodes. Further nodes can be connected either to a or b in order to retain frequencies.

Unfrequent unconnected patterns are built, starting from frequent (either connected or unconnected) patterns according to a similar strategy. Two randomly chosen frequent patterns p and q generate an unfrequent unconnected pattern r by connecting OUTBORDER(p) and INBORDER(q) with exactly one edge exhibiting a low frequency. Further nodes are added and connected either to OUTBORDER(p) or to INBORDER(q) in order to retain frequencies. The resulting graph r still has $f_r = \max(f_p, f_q)$, but $p \cup q$ has frequency 1.

The u and d parameters influence the number of frequent and unfrequent unconnected patterns to be generated. Starting from a set d of connected patterns, unconnected frequent patterns are generated until u patterns are obtained. These are used to iteratively generate unfrequent unconnected patterns, until a single graph is obtained. On the basis of this description, we can expect that, the larger the difference between d and u, the higher is the number of unconnected frequent patterns contained within synthesized data. On the other side, the lower is the difference, the higher is the number of unconnected unfrequent patterns. Notice also that the frequency of each unconnected frequent pattern is related to the number of unfrequent components and the number of desired total instances. Indeed, if u is the desired number of frequent unconnected patterns to compose infrequently, the number of instances $\mathcal{F}$ necessary to compose them with frequency at least f is $|\mathcal{F}| \approx u \times f$.

In a first set of experiments, we evaluated the ratio $f = n_{cc}/n_{cp}$ between the number n_{cc} of candidate patterns checked against the logs and the total number n_{cp} of candidate patterns. Low values of f represent a higher pruning capability of the algorithm *ws*-unconnected-find* w.r.t *ws-unconnected-find*. Figure 6(a) shows the behavior of f for $d = 10$, $minSupp = 5\%$ and increasing values of $\mathcal{F}$ and u. As we can see, f is quite low, except when $u = 8$. Figures 7(a)and 7(b) exhibit the number of unfrequent and frequent unconnected patterns discovered by resorting to upper and lower bounds, respectively.

Figure 6(b) exhibits the ratio f for increasing values of $minSupp$ and u, when $|\mathcal{F}| = 1.000$ and $d = 15$. Peaks within the graphs are mainly due to the fact that we are mining unconnected components: at low support values, patterns are mined as frequent connected (i.e., the frequency of paths connecting the components is greater than the given threshold). As soon as support threshold increases, paths are no more frequent, and hence a higher number of unconnected frequent patterns is detected by the algorithm. Despite of these irregularities, we can notice that increasing values of u influence the pruning ability. In particular, by figures 7(c) and 7(d) we can see that, with high values of u, upper bounds provide substantial pruning ability.

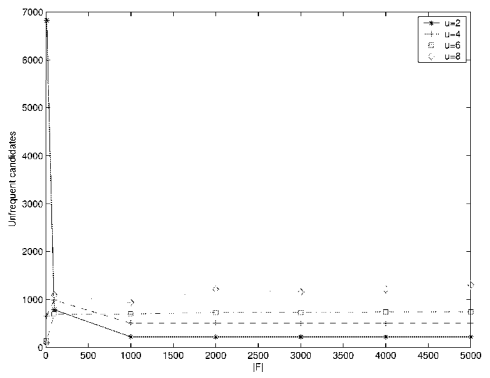
More in general, upper bounds are better in pruning, as also demonstrated by figures 8(a) and 8(b). In these graphs, the number of pruned unfrequent and frequent patterns is shown for increasing values of $minSupp$ and d, with u fixed to 2 and $\mathcal{F}$ to 1.000. Interestingly, lower bounds are quite effective at high values of $minSupp$, which guarantee several disconnections among frequent patterns.

As a final remark, it is worth mentioning that upper bounds tend to be effective in the first steps of the algorithm (i.e., in the computation of L_k for low values of k), whereas lower bounds effectiveness distributes throughout the entire execution of the algorithm. More extensive graphical analysis, omitted here for lack of space, gives evidence of the claim.

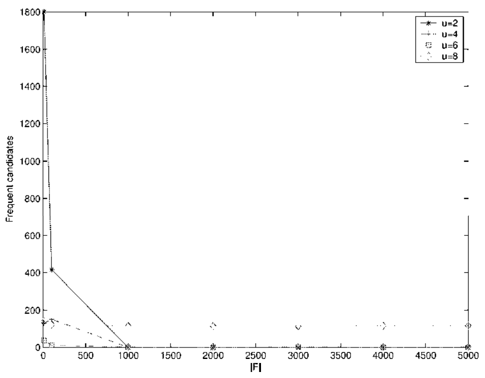
6 Conclusions

In this paper we have addressed the problem of mining frequent unconnected workflow patterns. We have developed a graph theoretic approach for predicting whether activities in a workflow are coupled in the executions, on the basis of the workflow structure and of the frequency of the elementary activities alone. The approach has been adopted in a level-wise algorithm for mining frequent patterns, and revealed as a powerful tool for pruning the search space of candidate patterns.

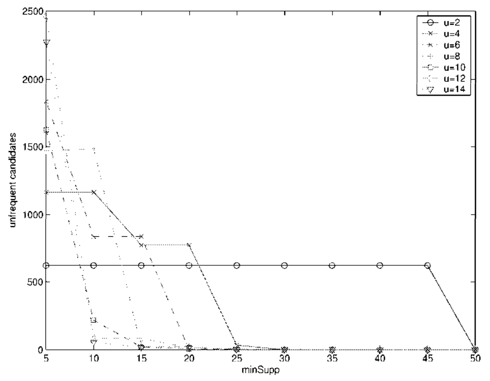
We conclude by sketching some directions of future research. The models proposed in this paper and in [8] are essentially *propositional* models, for they assume a simplification of the workflow schema in which many real-life details are omitted. However, we believe that the models can be easily updated to cope with more complex constraints, such as time constraints, pre-conditions



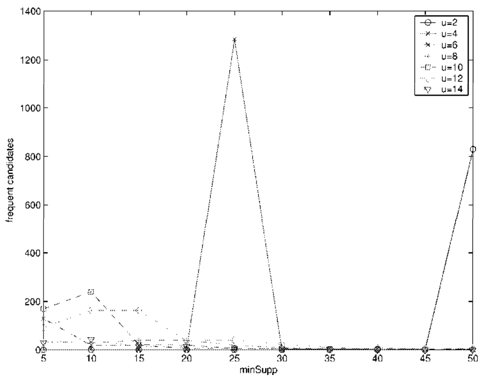
(a) Pruning by upper bound, for increasing values of $|\mathcal{F}|$ and u.



(b) Pruning by lower bound, for increasing values of $|\mathcal{F}|$ and u.



(c) Pruning by upper bound, for increasing values of *minSupp* and u.



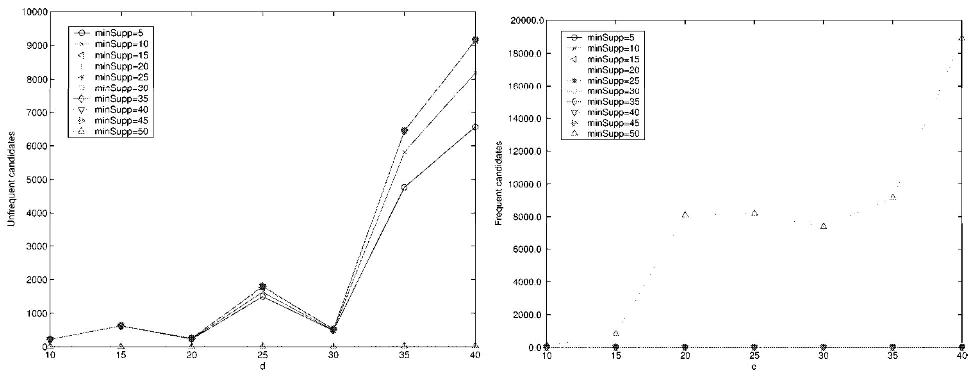
(d) Pruning by lower bound, for increasing values of *minSupp* and u.

Fig. 7. Performance Graphs (cont.).

and post-conditions, and rules for exception handling. Furthermore, we believe that many of the observations we exploit in the paper can be used for performing similar optimizations in different contexts in which the model of the data is assumed to be a graph [10, 11, 16].

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 Large Databases In *Proc. 20th Int. Conf. on Very Large Data Bases(VLDB'94)*, pages 487–499,1994.
3. A. Bonifati, F. Casati, U. Dayal, and M.-C. Shan. Warehousing Workflow Data: Challenges and Opportunities. In *Proc. 25th Int. Conf. on Very Large Data Bases (VLDB'01)*, pages 649–652, 2001.
4. J. E. Cook, and A. L. Wolf. Automating Process Discovery Through Event-Data Analysis. In *Proc. 17th Int. Conf. on Software Engineering (ICSE'95)*, pages 73–82, 1995.
5. 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 (VLDB'01)*, pages 3-13, 2001.
6. 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.



(a) Pruning by upper bound, for increasing values of d and $minSupp$.
(b) Pruning by lower bound, for increasing values of d and $minSupp$.

Fig. 8. Performance Graphs (cont.).

7. M. Gillmann, W. Wonner, and G. Weikum. Workflow Management with Service Quality Guarantees. In *Proc. ACM Conf. on Management of Data (SIGMOD'02)* , 2002.
8. Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, Domenico Saccà. Mining Frequent Instances on Workflows. *PAKDD 2003*, 2003, pp 209-221.
9. P. Koksal, S. N. Arpinar, and A. Dogac. Workflow History Management. *SIGMOD Record*, 27(1), pages 67–75, 1998.
10. M. Kuramochi, and G. Karypis. Frequent Subgraph Discovery. In *Proc. of the Int. Conf. on Data Mining (ICDM'01)*, pages 313-320, 2001.
11. 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.
12. P. Senkul, M. Kifer, and I.H. Toroslu. A logical Framework for Scheduling Workflows Under Resource Allocation Constraints. In*Proc. 28th Int. Conf. on Very Large Data Bases (VLDB'02)*, pages 694–702, 2002.
13. W. M. P. van der Aalst, and K. M. van Hee. Workflow Management: Models, Methods, and Systems. *MIT Press*, 2002.
14. W. M. P. van der Aalst, A. Hirnschall, and H.M.W. Verbeek. An Alternative Way to Analyze Workflow Graphs. In *Proc. of the 14th Int. Conf. on Advanced Information Systems Engineering*, pages 534–552, 2002.
15. 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*, to appear.
16. M. Zaki. Efficiently Mining Frequent Trees in a Forest. In *Proc. 8th Int. Conf. On Knowledge Discovery and Data Mining (SIGKDD'02)*, 2002. to appear.