Universiteit Antwerpen

Technical Report

09.02

Mining Interesting Sets and Rules in Relational Databases

Bart Goethals, Wim Le Page, Michael Mampaey

University of Antwerp
Department of Mathematics and Computer Science
Middelheimlaan 1
B-2020 Antwerp – Belgium

Abstract

In this paper we propose a new and elegant approach toward the generalization of frequent itemset mining to the multi-relational case. We define relational itemsets that contain items from several relations, and a support measure that can easily be interpreted based on the key dependencies as defined in the relational scheme. We present an efficient depth-first algorithm, which mines relational itemsets directly from arbitrary relational databases. Several experiments show the practicality and usefulness of the proposed approach. This technical report is an extended version of our work published in [10]

1. Introduction

Itemset mining algorithms are probably the most well-known algorithms in the field of frequent pattern mining. Many efficient solutions have been developed for this relatively simple class of patterns. These solutions can be applied to relational databases containing a *single* relation. While the task of mining frequent itemsets in a single relation is well-studied, for mining frequent itemsets in arbitrary relational databases which typically have *more* than one relation, only a few solutions exist [5,6,12,13]. These methods consider a *relational itemset* to be a set of items, where each item is an attribute-value pair, and these items belong to one or more *relations* in the database. In order for two such items from different relations to be in the same itemset, they must be *connected.* Two items (attribute-value pairs) are considered to be connected if there exists a join of their two relations in the database that connects them. In general, an itemset is said to *occur* in the database, if there exists a tuple in a join of the relations that contains the itemset. In this paper we also adopt this notion of occurrence.

A good definition of a unit in which the support of a pattern is expressed — i.e. what is being counted — is a primary requirement to mine any kind of frequent pattern. In standard itemset mining there is a single table of transactions and the unit to be counted is clearly defined as the number of transactions containing the itemset. For example, a frequent itemset {*butter,cheese*} represents the fact that in a large fraction of the transactions *butter* and *cheese* occur together. When considering itemsets over multiple relations, elegantly defining such a unit is less obvious. In the existing relational itemset mining approaches [5,12,13], the frequency of an itemset over multiple relations is expressed in the number of occurrences (tuples) in the result of a join of the relations in the database. However, this definition of itemset frequency is hard to interpret as it heavily depends on how well the items in the set are connected. For instance, in a relational database consisting of the tables Customer, Product and Supplier, the singleton itemset {*butter*} could occur frequently in the join because it is connected to a lot of *customers*, but even if this is not the case, it could still be frequent simply because it is connected to a lot of *suppliers*. The main issue here is that using the number of occurrences in the join as a support measure, it is hard to determine the true *cause* of the frequency.

Instead of counting frequency as the number of occurrences in the join, we opt for a novel, more semantic approach. In the supermarket example, we want to find products that are bought by a lot of (different) customers, but we might also be interested in products that have a lot of (different) suppliers. Essentially, we want to count the number of connected customers and the number of connected suppliers. In order to have such semantics, we must not count all occurrences of an itemset in the join, but instead separately count the occurrences of unique customers and the occurrences of unique suppliers. This new frequency counting technique allows for interpretable frequent itemsets, since we will now have separate customer-frequent and supplier-frequent itemsets.

As our goal is to mine relational itemsets in arbitrary relational databases, we assume that key dependencies are specified in the relational scheme of the input database. This way, we determine the support of an itemset by counting unique key values in the tuples where the itemset occurs. Consider the Entity-Relationship scheme in Figure 1, which we will use as a running example throughout the paper. For this scheme, the keys to be used are {Professor.PID, Course.CID, Student.SID, Study.YID}. It goes without saying that itemsets frequent in Professor.PID have different semantics than itemsets frequent in Course.CID. It is clear that our new

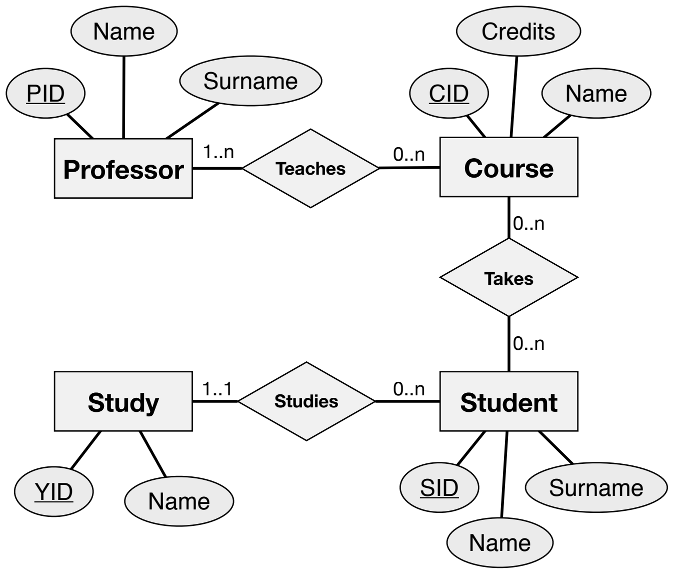


FIGURE 1. Running Example of a Relational Scheme.

frequency counting method significantly differs from other relational itemset mining approaches that simply count the occurrences in the join of all relations. As we show in this paper, our new approach permits an efficient propagation-based depth-first algorithm that generates interesting frequent relational itemsets that are sensible and easy to interpret. Since we want to provide a pattern mining algorithm that can directly interact with standard *relational database management systems*, we use SQL in our implementation to fetch the necessary data from the database. As a result, our algorithm is generally and easily applicable to arbitrary relational databases, without requiring any transformation on the data.

2. Definitions

Before formally defining relational itemsets, we first consider the relational scheme as it forms the basis of our definition of patterns.

2.1. Relational Scheme.

Every relational database has a relational scheme. For the sake of clarity, we focus on simple relational schemes. More specifically, we consider acyclic relational schemes using only binary relations to connect entities, i.e. schemes that can be represented as an unrooted tree (such as the scheme in Figure 1). Let *sort* be a function that maps a relation name to its attributes [1]. We now define such schemes as follows.

Definition 1. *Let $\mathcal{E}$ be a set of entities and $\mathcal{R}$ a set of binary relations. A* ***simple relational scheme*** *is a tuple $(\mathcal{E}, \mathcal{R})$ such that*

(1) $\forall E \in \mathcal{E} : \exists! key(E) \subseteq sort(E)$*, the key attributes of* E
(2) $\forall R \in \mathcal{R} : \exists! E_i, E_j \in \mathcal{E}, E_i \neq E_j$ *such that* $sort(R) = key(E_i) \cup key(E_j)$
(3) $\forall E_i, E_j \in \mathcal{E}, E_i \neq E_j : \exists! E_1, \ldots, E_n \in \mathcal{E}$ *such that*
  (a) $E_1 = E_i$ *and* $E_n = E_j$
  (b) $\forall k, l :$ *if* $k \neq l$ *then* $E_k \neq E_l$
  (c) $\forall k, \exists! R \in \mathcal{R} : sort(R) = key(E_k) \cup key(E_{k+1})$

Informally, this states that every entity needs to have a unique key defined, and that for every two entities there exists a unique path of binary relations and entities connecting them. Many realistic relational databases satisfy such simple relational schemes. Moreover, databases with star schemes and snowflake schemes can be formulated in this way.

2.2. Relational Itemsets.

We are now ready to formally define the type of pattern we want to mine.

Definition 2. *Given a simple relational scheme $(\mathcal{E}, \mathcal{R})$, the set $\{(A_1 = v_1), \ldots, (A_n = v_n)\}_K$ is a* ***relational itemset*** *in key K where $K \in \bigcup_{E \in \mathcal{E}} \{key(E)\}$, each $(A_i = v_i)$ is an attribute-value pair (or item) such that $A_i \in \bigcup_{E \in \mathcal{E}} sort(E)$ and $v_i \in Dom(A_i)$, and there is no $(A_j = v_j)$ with $j \neq i$ such that $A_j = A_i$. We denote the set of all items by $\mathcal{I}$.*

Note, that we will sometimes use I to denote a set of items without any key specified, next to I_K which denotes an itemset in key K.

Example 1. *Given the relational scheme in Figure 1, let us abbreviate the relation names to P, C, Stnt and Stdy. We can consider the relational itemset $\{(P.Name = Jan), (C.Credits = 10)\}_{C.ID}$. This itemset describes courses that have 10 credits and that are taught by a professors named Jan. Since we have C.ID as the key we know we are expressing patterns about courses. Because we assume a simple relational scheme, the professor can only be connected to the courses in one way (via the relation Teaches) and hence, we know this pattern expresses that Jan is teaching them.*

Next, we define the *support* measure for relational itemsets. In order to do this we need to consider the unique path of entities and relations connecting the entities of an itemset.

Proposition 1. *Given a simple relational scheme* $(\mathcal{E}, \mathcal{R})$*, and a relational itemset* $I_K = \{(A_1 = v_1), \ldots, (A_n = v_n)\}_K$*, let* $\mathcal{E}_{I_K} = \{E \in \mathcal{E} \mid \mathrm{key}(E) = K \text{ or } \exists i : A_i \in sort(E)\}$*. There exists a* ***unique path*** P_{I_K} *connecting all entities* $E \in \mathcal{E}_{I_K}$*.*

Proof. (outline) We can consider the simple relational scheme $(\mathcal{E}, \mathcal{R})$ to be a tree where $\mathcal{E}$ are the nodes, and $\mathcal{R}$ are the edges. Then, we can consider the subtree formed by the nodes in $\bigcup_{E_i, E_j \in \mathcal{E}_{I_K}} \mathrm{path}(E_i, E_j)$ and the edges in $\mathcal{R}$ connecting them, where $\mathrm{path}(E_i, E_j)$ is the unique path of entities as defined in Definition 1. If we take $E_K : \mathrm{key}(E_K) = K$ to be the root of this tree, we can then consider the preorder traversal of this subtree as the unique path P_{I_K}. □

Definition 3. *Given an instance of a simple relational scheme* $(\mathcal{E}, \mathcal{R})$*, the* ***absolute support*** *of a relational itemset* $I_K = \{(A_1 = v_1), \ldots, (A_n = v_n)\}_K$ *is the number of distinct values of the key* K *in the answer of the query (expressed here in relational algebra [1]):*

$$\pi_K \sigma_{A_1 = v_1, \ldots, A_n = v_n} E_1 \bowtie_{key(E_1)} R_{1,2} \bowtie_{key(E_2)} E_2 \bowtie \ldots \bowtie E_m$$

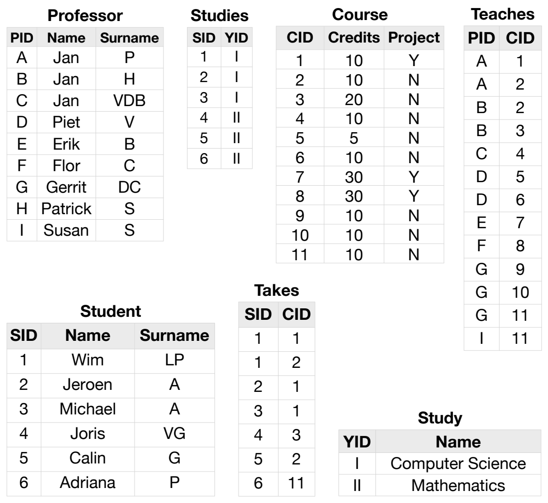
where $E_i \bowtie_{\mathrm{key}(E_i)} R_{i,i+1}$ *represents the equi-join on* $E_i.\mathrm{key}(E_i) = R_{i,j}.\mathrm{key}(E_i)$ *and all* $E_i \in \mathcal{E}_{I_K}$ *are joined using the unique path* P_{I_K}*. We call the result of this query the* ***KeyID list*** *of* I_K*. The* ***relative support*** *of an itemset is the absolute support of that itemset divided by the number of distinct values of* K *in the entity* E *of which* K *is key, i.e. the number of tuples in* E*. We call a relational itemset* ***frequent*** *if its (absolute/relative) support exceeds a given* ***minimal (absolute/relative) support threshold****.*

Example 2. *Given Figure 2a, which is an instance of the scheme in Figure 1, the absolute support of the itemset* $\{(\mathsf{C.Credits} = \mathsf{10})\}_{\mathsf{P.PID}}$ *is 6, as the answer to the query* $\pi_{\mathsf{P.PID}} \sigma_{\mathsf{C.Credits=10}} \mathsf{P} \bowtie_{\mathsf{PID}} \mathsf{Teaches} \bowtie_{\mathsf{CID}} \mathsf{C}$ *is* $\{\mathtt{A,B,C,D,G,I}\}$*. This means that six professors teach a course with ten credits. The relative support is 6/9, as there are nine professors in the* Professor *relation.*

Proposition 2. *The support measure given in Definition 3 is monotonic with respect to set inclusion, i.e. for all keys* K *and for all itemsets* I_K, J_K *it holds that*

$$I_K \subseteq J_K \Rightarrow supp(I_K) \geq supp(J_K).$$

Proof. (sketch) Denote by $\bowtie_{I_K}$ the join of all entities and relations in $\mathcal{E}_{I_K}$, and similarly, let $\bowtie_{J_K}$ be the join of all entities and relations in $\mathcal{E}_{J_K}$. It is clear that $\mathcal{E}_{\mathcal{I}_K} \subseteq \mathcal{E}_{\mathcal{J}_K}$. The number of unique key values of K in $\bowtie_{J_K}$ cannot be greater than the number of unique key values in $\bowtie_{I_K}$. Furthermore, the select statement σ_{J_K} can only exclude more key values when projected to K, than the selection σ_{I_K}. □



Professor

PID	Name	Surname
A	Jan	P
B	Jan	H
C	Jan	VDB
D	Piet	V
E	Erik	B
F	Flor	C
G	Gerrit	DC
H	Patrick	S
I	Susan	S

Studies

SID	YID
1	I
2	I
3	I
4	II
5	II
6	II

Course

CID	Credits	Project
1	10	Y
2	10	N
3	20	N
4	10	N
5	5	N
6	10	N
7	30	Y
8	30	Y
9	10	N
10	10	N
11	10	N

Teaches

PID	CID
A	1
A	2
B	2
B	3
C	4
D	5
D	6
E	7
F	8
G	9
G	10
G	11
I	11

Student

SID	Name	Surname
1	Wim	LP
2	Jeroen	A
3	Michael	A
4	Joris	VG
5	Calin	G
6	Adriana	P

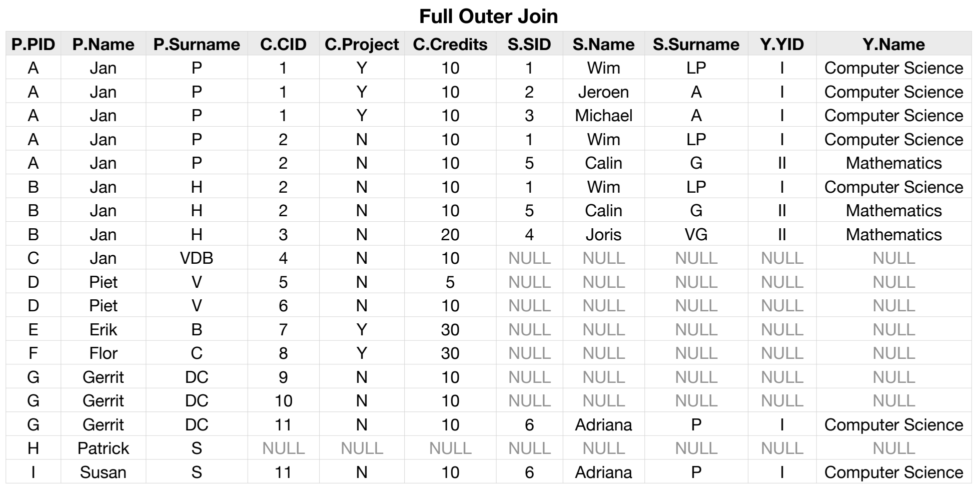
Takes

SID	CID
1	1
1	2
2	1
3	1
4	3
5	2
6	11

Study

YID	Name
I	Computer Science
II	Mathematics

(A) Example relational instance



Full Outer Join

P.PID	P.Name	P.Surname	C.CID	C.Project	C.Credits	S.SID	S.Name	S.Surname	Y.YID	Y.Name
A	Jan	P	1	Y	10	1	Wim	LP	I	Computer Science
A	Jan	P	1	Y	10	2	Jeroen	A	I	Computer Science
A	Jan	P	1	Y	10	3	Michael	A	I	Computer Science
A	Jan	P	2	N	10	1	Wim	LP	I	Computer Science
A	Jan	P	2	N	10	5	Calin	G	II	Mathematics
B	Jan	H	2	N	10	1	Wim	LP	I	Computer Science
B	Jan	H	2	N	10	5	Calin	G	II	Mathematics
B	Jan	H	3	N	20	4	Joris	VG	II	Mathematics
C	Jan	VDB	4	N	10	NULL	NULL	NULL	NULL	NULL
D	Piet	V	5	N	5	NULL	NULL	NULL	NULL	NULL
D	Piet	V	6	N	10	NULL	NULL	NULL	NULL	NULL
E	Erik	B	7	Y	30	NULL	NULL	NULL	NULL	NULL
F	Flor	C	8	Y	30	NULL	NULL	NULL	NULL	NULL
G	Gerrit	DC	9	N	10	NULL	NULL	NULL	NULL	NULL
G	Gerrit	DC	10	N	10	NULL	NULL	NULL	NULL	NULL
G	Gerrit	DC	11	N	10	6	Adriana	P	I	Computer Science
H	Patrick	S	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
I	Susan	S	11	N	10	6	Adriana	P	I	Computer Science

(B) Full outer join of the relational instance

FIGURE 2. Instance of the Relational Scheme in Figure 1.

2.3. **Relational Association Rules.** Association rules are defined in the same way as in standard itemset mining. The only restriction is that the antecedent and the consequent need to be expressed in the same key.

Definition 4. *Let $\mathcal{I}_K$ be the set of all relational items in key K. The rule $A \Rightarrow_K C$ is a* ***relational association rule*** *in key K if $A, A \cup C \subseteq \mathcal{I}_K$.*

Definition 5. *The* ***support*** *of $A \Rightarrow_K C$ is the support of $(A \cup C)_K$. The* ***confidence*** *of $A \Rightarrow_K C$ is the support of $(A \cup C)_K$ divided by the support of A_K.*

Example 3. *Given the instance depicted in Figure 2a, an example of a relational association rule is* $\{(\mathsf{P.Name} = \mathtt{Jan})\} \Rightarrow_{\mathsf{C.CID}} \{(\mathsf{C.Credits} = \mathtt{10})\}$*. The confidence is* $3/4 = 0.75$ *since there are three courses* $\{\mathtt{1,2,4}\}$ *taught by a '*`Jan`*' that have ten credits, compared to the four courses* $\{\mathtt{1,2,3,4}\}$ *taught by a '*`Jan`*' in total. The relative support is* $3/11 = 0.27$ *since there are eleven courses in total.*

3. Algorithm

In this section we present two algorithms for mining relational itemsets. We first construct a naive algorithm based on the computation of the full outer join. Then we present our SMuRFIG algorithm (**S**imple **Mu**lti-**R**elational **F**requent **I**temset **G**enerator). Both algorithms employ KeyID lists, similar to the *tid* (transaction identifier) lists used in the well-known Eclat algorithm [16].

3.1. **Naive.**

First, we consider the naive approach. The input of the Naive algorithm (see Algorithm 1) is an instance of a simple relational scheme and a relative support threshold *minsup*. The support query from Definition 3 is straightforwardly decomposed into three parts, i.e. a join, a selection, and a projection. First, a join table J is constructed, in which the correct supports are to be found. However, this join is different for each itemset, and performing all such possible joins is infeasible. Instead, we create a single large join table using all entities and relations. To construct J, we cannot use an equi-join. Indeed, if a tuple is not connected to any tuples in other tables, it does not appear in the full equi-join of all entity tables, which means we lose some information. To avoid this, we combine the entities and relations using a *full outer join*, which combines all non-connected tuples with `NULL`-values. This full outer join results in table J on Line 1.

Example 4. *The result of the full outer join of the entities and relations of Figure 2a is given in Figure 2b. Here, there are* `NULL` *values present for the student attributes on the rows for courses 4 to 10, since no students are taking them.*

Next, a standard frequent set Miner, Eclat, is applied to J using a new threshold (Line 3). Rather than using the relative support threshold *minsup*, we must use a new absolute threshold *abssup* (Line 2) for J. The absolute support of an itemset in J is at least as high as the absolute support of that itemset for any key K, so any itemset frequent in some key K with respect to *minsup* will also be frequent in J with respect to *abssup*. In a way, the Eclat algorithm fulfills the role of the select clause (σ) of the support query. We assume that the tid lists of the itemsets (the lists of tuples of J where the itemsets occur) are part of the output of Eclat. Finally, these tid lists are *translated* to their appropriate KeyID lists on Line 6. Translating a tid list T to a KeyID list is comes down to performing the projection $\pi_K(J \bowtie T)$ to each key K. This can be done efficiently by using lookup tables which can be created during the construction of J. At the end, the relative minimum support threshold is imposed for each itemset's KeyID list, and the frequent itemsets are reported.

Algorithm 1 Naive: straightforward frequent itemset miner

```
Input: An instance of a simple relational scheme $(\mathcal{E}, \mathcal{R})$; a relative support threshold $minsup$
Output: Set $\mathcal{F}$ of all frequent itemsets $I$
1: $J := E_1 \bowtie R_{1,2} \bowtie E_2 \bowtie \ldots \bowtie E_n$
2: abssup := $\min_{E \in \mathcal{E}}(minsup \times |E|)$
3: $\mathcal{I}_t$ := Eclat($J$, abssup)
4: for all $I_t \in \mathcal{I}_t$ do
5:   for all $E \in \mathcal{E}$ do
6:     KeyIDs($I_{key(E)}$) := translate($I_t$,key($E$))
7:     if supp($I_{key(E)}$) $\geq minsup \times |E|$ then
8:       $\mathcal{F} := \mathcal{F} \cup I_{key(E)}$
9: return $\mathcal{F}$
```

Example 5. *The itemset* $\{(\textsf{C.Credits} = \texttt{10})\}$ *corresponds to the transaction ids* $\{$`1,2,3,4,5,6,7,9,11,14,15,16,18`$\}$ *of the full outer join shown in Figure 2b. If we translate these to unique* P.PID*s by looking up the* P.PID *values for these tuple ids in the join table, we get* $\{$`A,B,C,D,G,I`$\}$*, and hence the absolute support of itemset* $\{(\textsf{C.Credits} = \texttt{10})\}_{\textsf{P.PID}}$ *is* 6*, corresponding to the result in Example 2.*

The advantage of the Naive algorithm is that it can be implemented as a wrapper around an existing itemset mining algorithm. However, there are several drawbacks to this method. First of all, the computation of the full outer join is costly with respect to both computation time and memory consumption, making it infeasible to use on larger databases. Secondly, too many candidates are generated. We can only prune itemsets that are infrequent in J with respect to *abssup*, but many candidate itemsets that are frequent in J may turn out to be infrequent for all entity keys K with respect to the *minsup* threshold.

3.2. **SMuRFIG.** The SMuRFIG algorithm (see Algorithm 2) does not suffer from the disadvantages of the Naive algorithm, i.e. it is efficient in both time and memory. It uses the concept of *KeyID list propagation.* First, the KeyID lists of all singletons are fetched from the data in their respective entities (Line 3), and then these KeyID lists are propagated to all other entities (Line 5, see Algorithm 3). The propagation function recursively translates a KeyID list from one entity E_i to its adjacent entities E_j, until all entities are reached. The translation of a KeyID list T_i from E_i to E_j via $R_{i,j}$ is equivalent to the result of the relational query $\pi_{key(E_j)}(T_i \bowtie R_{i,j})$. It is not difficult to verify that in this way we now have the KeyID lists of all singleton itemsets for all entity keys K, and hence their supports.

Next, the (frequent) singleton itemsets are combined into larger sets by the *Keyclat* function (Algorithm 4), which is the core of SMuRFIG. The search space is traversed depth-first. In each recursion, two k-itemsets I' and I'' with a common prefix P (initially empty) are combined to form a new candidate set $I = I' \cup I''$

Algorithm 2 SMuRFIG: Simple Multi-Relational Frequent Itemset Generator

```
Input: An instance of a simple relational scheme $(\mathcal{E}, \mathcal{R})$; a relative minimum
    support threshold $minsup$
Output: Set $\mathcal{F}$ of all frequent itemsets $I$
1: for all $E \in \mathcal{E}$ do
2:   $K := key(E)$
3:   $\mathcal{I}_E :=$ singletons($E$)
4:   for all $I_K \in \mathcal{I}_E$ do
5:     Propagate(KeyIDs($I_K$))
6:     if $\exists K' : \text{supp}(I_{K'}) \geq minsup \times |E|$ then
7:       $\mathcal{I} := \mathcal{I} \cup \{I\}$
8: $\mathcal{F} :=$ Keyclat($\mathcal{I}$, $minsup$)
9: return $\mathcal{F}$
```

Algorithm 3 Propagate: KeyID list propagation

```
Input: KeyID list of an itemset $I$ for key $K$ of some entity $E_i$
Output: KeyID lists of $I$ in all keys $K$ for all $E \in \mathcal{E}$
1: for all neighbours $E_j$ of $E_i$ not yet visited do
2:   Translate KeyIDs in $key(E_i)$ to $key(E_j)$ via $R_{i,j}$
3:   Propagate(KeyIDs in $key(E_j)$)
```

of size $k+1$. To compute the support of I for all keys, we first determine that the entity tables of the suffixes of I' and I'' are E_1 and E_2 (Line 4). We then intersect the KeyID lists of I' and I'' in E_1 and E_2, to obtain the support of I in E_1 and E_2, and then these KeyID lists are propagated to all other entities (Line 8). For these remaining entities E with key K, it does not necessarily hold that KeyIDs(I'_K) $\cap$ KeyIDs(I''_K) results in KeyIDs(I_K). We must additionally intersect this with the propagated KeyID lists of I_K in E, in order to obtain the final KeyID list of I_K in E (Line 11).

Example 6. *Suppose that we have the two itemsets $P = \{(\mathsf{P.Name} = \mathsf{Jan})\}$ and $S = \{(\mathsf{S.Surname} = \mathsf{A})\}$. These are singletons, so we have computed the KeyID lists for all keys. Furthermore, their common prefix is the empty set. We determine the entity table of P to be* Professor *and that of S to be* Student*. Let PS denote $P \cup S$, then we compute*

$$\begin{aligned}
\text{KeyIDs}(PS_{\mathsf{P.PID}}) &= \text{KeyIDs}(P_{\mathsf{P.PID}}) \cap \text{KeyIDs}(S_{\mathsf{P.PID}}) \\
&= \{\mathtt{A}, \mathtt{B}, \mathtt{C}\} \cap \{\mathtt{A}\} = \{\mathtt{A}\} \\
\text{KeyIDs}(PS_{\mathsf{S.SID}}) &= \text{KeyIDs}(P_{\mathsf{S.SID}}) \cap \text{KeyIDs}(S_{\mathsf{S.SID}}) \\
&= \{1, 2, 3, 4, 5\} \cap \{2, 3\} = \{2, 3\}
\end{aligned}$$

Algorithm 4 Keyclat: computation of itemsets' KeyID lists

```
Input: Set of $k$-itemsets $\mathcal{L}_P$ with a common prefix $P$; relative minimum support
    threshold $minsup$
Output: Set $\mathcal{F}$ of frequent itemsets $I$ with prefix $P$
 1: for $I'$ in $\mathcal{L}_P$ do
 2:   for $I''$ in $\mathcal{L}_P$ with $I'' > I'$ do
 3:     $I := I' \cup I''$
 4:     $E_1, E_2 :=$ entity of suffix items $I' \setminus P, I'' \setminus P$ resp.
 5:     for $i \in \{1,2\}$ do
 6:       $K_i := key(E_i)$
 7:       $\text{KeyIDs}(I_{K_i}) := \text{KeyIDs}(I'_{K_i}) \cap \text{KeyIDs}(I''_{K_i})$
 8:       $\text{pKeyIDs}_i(I) := \text{Propagate}(\text{KeyIDs}(I_{K_i}))$
 9:     for $E \in \mathcal{E} \setminus \{E_1, E_2\}$ do
10:       $K := key(E)$
11:       $\text{KeyIDs}(I_K) := \text{pKeyIDs}_1(I_K) \cap \text{pKeyIDs}_2(I_K)$
                         $\cap\ \text{KeyIDs}(I'_K) \cap \text{KeyIDs}(I''_K)$
12:       $\text{supp}(I_K) := |\text{KeyIDs}(I_K)|$
13:       if $\text{supp}(I_K) \geq minsup \times |E|$ then
14:         $\mathcal{F}_{I'} := \mathcal{F}_{I'} \cup I_K$
15:   $\mathcal{F} := \mathcal{F} \cup \mathcal{F}_{I'} \cup \text{Keyclat}(\mathcal{F}_{I'}, minsup)$
16: return $\mathcal{F}$
```

Then these KeyID lists are propagated to all entities. For example, propagating the P.PIDs $\{\mathtt{A}\}$ *to C.CID results in* $\text{pKeyIDs}_1(PS_{\mathsf{C.CID}}) = \{1, 2\}$*. Propagating S.SIDs* $\{2, 3\}$ *results in* $\text{pKeyIDs}_2(PS_{\mathsf{C.CID}}) = \{1\}$*. To obtain the other KeyID lists we perform intersections. For C.CID this becomes*

$$\begin{aligned}\text{KeyIDs}(PS_{\mathsf{C.CID}}) &= \text{pKeyIDs}_1(PS_{\mathsf{C.CID}}) \cap \text{pKeyIDs}_2(PS_{\mathsf{C.CID}}) \\ &\quad \cap \text{KeyIDs}(P_{\mathsf{C.CID}}) \cap \text{KeyIDs}(S_{\mathsf{C.CID}}) \\ &= \{1, 2\} \cap \{1\} \cap \{1, 2\} \cap \{1\} = \{1\}\end{aligned}$$

Thus, the support of $\{(\mathit{P.Name} = \mathit{Jan}), (\mathit{S.Surname} = \mathit{A})\}_{\mathit{C.CID}} = |\{1\}| = 1$*. It is clear that this result corresponds to what can be seen in the full outer join in Figure 2b.*

The time complexity of SMuRFIG is as follows. Per itemset at most three intersections are required for each entity E, taking $O(\sum_{E \in \mathcal{E}} |E|)$, where $|E|$ is the number of tuples in E. The propagation function is executed at most twice and uses each relation R once, amounting to $O(\sum_{R \in \mathcal{R}} |R|)$. Hence, the time complexity of SMuRFIG is $O\left(|\mathcal{F}| \cdot size(\mathcal{DB})\right)$, where $|\mathcal{F}|$ is the total number of frequent itemsets and $size(\mathcal{DB}) = \sum_{E \in \mathcal{E}} |E| + \sum_{R \in \mathcal{R}} |R|$ is the size of the database.

SMuRFIG only requires a small amount of patterns to be stored in memory simultaneously. When an itemset of length k is generated, we have $(k^2 + k)/2$

previous itemsets in memory due to the depth-first traversal. For each of these itemsets, KeyID lists are stored for all keys. The maximal total size of these lists for one itemset is $\sum_{E\in\mathcal{E}} |E|$. Next to this, we also keep all relations R in memory, which are needed in the propagation function. To sum up, if l is the size of the largest frequent itemset, then SMuRFIG's worst case memory consumption is $O\left(l^2 \cdot \sum_{E\in\mathcal{E}} |E| + \sum_{R\in\mathcal{R}} |R|\right)$. Note, however, that in practice the size of a KeyID list can be much smaller than the corresponding entity $|E|$.

We can implement some optimizations in SMuRFIG. For this, we need to know the shape of the schema graph. We say that an entity E lies *between* two entities E_1 and E_2 if the unique path from E to E_1 does not contain the path from E to E_2 or vice versa. If E does not lie between E_1 en E_2, then E will always be closer to one of them. Then, on Line 11 of Algorithm 4, we can reduce the number of intersections by observing that if E lies between E_1 and E_1 then $\text{KeyIDs}(I_K) := \text{KeyIDs}(I'_K) \cap \text{KeyIDs}(I''_K)$. If E does not lie between E_1 and E_2 and it is closest to, say, E_1, then $\text{KeyIDs}(I_K) := \text{KeyIDs}(I'_K) \cap \text{KeyIDs}(I''_K) \cap \text{pKeyIDs}_1(I_K)$. This reduces the number of required computations on Line 11. Note that determining which entity lies between which other entities can be done once before the algorithm starts, and then this information can be stored in a lookup table.

4. Support Deviation

The relational setting that we are considering also brings additional challenges. For instance, let us consider the itemset $\{(\mathsf{C.project} = \mathtt{Y})\}_{\mathsf{S.SID}}$ having a support of 67%. For a lower support threshold, this would be a frequent itemset, but it is not necessarily an *interesting* one. Suppose that we also find that $\{(\mathsf{C.project} = \mathtt{Y})\}$ holds for 30% of the courses. Depending on the connectedness of students and courses, a support of 67% could even be the expected value. For example, if students typically take one course then the expected support (if one assumes no bias) would be 30%. However, if they take two courses each it rises to 51.2%, for three courses this becomes 66.1%, etc. So in the case of an average of three courses per student, a support of 67% is expected, and thus we could consider this pattern to be uninteresting. Hence, in order to determine if $\{(\mathsf{C.project} = \mathtt{Y})\}_{\mathsf{S.SID}}$ is interesting, we use the connections and the support in $\mathsf{C.CID}$ to compute the *expected support* of $\{(\mathsf{C.project} = \mathtt{Y})\}_{\mathsf{S.SID}}$, and we discard the itemset if its real support does not deviate substantially from its expected support.

To formalize this notion, we start off by only considering *intra-entity* itemsets, i.e. relational itemsets I consisting of items from a single entity E with key K. We only want to find those frequent itemsets $I_{K'}$ where the support in K' deviates enough from the expected support in K'. We now formally define expected support in our context (based on the general definition of expected value) as follows.

Definition 6. *Let $I \subseteq \mathcal{I}$ be an intra-entity itemset containing items from a single relation E with key K, and let K' be the key of some other entity E'. Furthermore, let S and S' be two random variables for the support of I_K and $I_{K'}$, respectively. Then the* ***expected absolute support*** *of $I_{K'}$, given that supp(I_K) $= s$, equals*

$$E[S'|S = s] = \sum_{i=1}^{k'} \left(1 - \prod_{j=0}^{d_i - 1} \left(1 - \frac{s}{k - j} \right) \right)$$

where $k = |E|$, $k' = |E'|$, and d_i is the degree *of the i-th tuple in E', i.e. the number of tuples in E that tuple i is connected to. Note that when $k - d_i < s$ then $\prod_{j=0}^{d_i-1}(1 - \frac{s}{k-j}) = 0$.*

The formula above is derived as follows. The relative support s/k is equal to the probability $P(I)$ of a tuple in E. The probability that a connection of a tuple of E' goes to a tuple of E where I does *not* hold is $(1 - s/k)$. The probability that a second connection of a tuple of E' goes to a tuple of E where I does not hold, given that there already was a first connection that did not hold is $(1 - \frac{s}{k-1})$. Since for tuple i of E' there are d_i connections to tuples of E, the probability that none of them are to a tuple where I holds is $\prod_{j=0}^{d_i-1}(1 - \frac{s}{k-j})$, and therefore the probability that one of them *does* connect is $1 - \prod_{j=0}^{d_i-1}(1 - \frac{s}{k-j})$. We then take the sum over all tuples i of E' and obtain the stated formula. Using this definition of expected support, we formally introduce *deviation.*

Definition 7. *Given an itemset $I \subseteq \mathcal{I}$, let E_K be the entity of key K. We define the* ***deviation*** *of I_K as*

$$\frac{|sup(I_K) - E[sup(I_K)]|}{|E_K|}.$$

Our goal is to find frequent relational itemsets with a given minimal deviation, in order to eliminate redundant relational itemsets. An experimental evaluation can be found in Section 6. Note that we restricted ourselves to a special case of intra-entity itemsets, i.e. itemsets only containing items from the same entity. In order to generalize deviation and expected support to itemsets containing attributes from multiple entities, which we refer to as *inter-entity* itemsets, it is necessary to generalize the definition of itemset support to allow sets of keys. This would entail substantial changes to our algorithm and is therefore part of our future work.

5. Redundancy

Examining the initial results of our algorithm, we uncovered several types of redundancies. In this section we generalize some popular existing redundancy measures to relational association rules. The types of redundancies encountered in standard association rule mining are also present here, and are worse due to the relational multiplication factor (the different keys and the different relations).

We extend the notion of closed itemsets [15] to our multi-relational setting.

Definition 8. *An itemset I is called* ***closed*** *if for all supersets I' of I there exists a key K such that $supp(I'_K) < supp(I_K)$. The* ***closure*** *of an itemset is defined as its smallest closed superset.*

Assume that we have a non-closed itemset A, where $supp(A_K) = supp(AB_K)$ for all keys K. It follows that the rule $A \Rightarrow B$ has a confidence of 100%. For any other association rule $A \Rightarrow_K C$ we see then that both its support and confidence are the same as that of $A \Rightarrow_K BC$ and $AB \Rightarrow_K C$, and hence these two rules are redundant, since they can be inferred from the first one. We will employ the same closure-based redundancy techniques as proposed by Zaki [15] to remove such rules. However, we must keep in mind that unlike in the single-table context, not every 100% rule entails a redundancy. For instance, if $A \Rightarrow_K B$ has a confidence of 100% for some but not all keys K, then for another key K' it might hold that the confidence of $AB \Rightarrow_{K'} C$ is different from that of $A \Rightarrow_{K'} C$. Next to the fact that we do not generate closure-based redundant rules, we also do not generate the 100% rules that follow from the known key dependencies of the scheme i.e., rules of the form $key(E) \Rightarrow A$ where $A \subseteq sort(E)$, since these rules are trivial.

Next to the lossless closure-based redundancy removal, we introduce an additional, but lossy, pruning technique. This technique is a generalization of minimal improvement [3]. In minimal improvement, all rules $A \Rightarrow C$ must have a confidence that is greater than any of its proper subrules $B \Rightarrow C$ with $B \subseteq A$, by at least some threshold. The aim of this measure is to eliminate rules $A \Rightarrow C$ that have more constraints than a simpler rule $B \Rightarrow C$, but that do not differ much in confidence. Minimal improvement, as the name implies, only allows rules to improve in confidence. While this makes sense in a market-basket analysis setting, a 'negative improvement' can also be interesting. Suppose that the rule $\{(\mathsf{Stdy.YID{=}I})\} \Rightarrow_{\mathsf{S.SID}} \{(\mathsf{C.Credits{=}10})\}$ has a confidence of 40%, telling us that 40% of the students of study `I` take a course with `10` credits. Suppose now we also have the rule $\{\} \Rightarrow_{\mathsf{S.SID}} \{(\mathsf{C.Credits{=}10})\}$ which has 90% confidence. This would imply that the students of the study `I` significantly diverge from the general population of all students, which could be an interesting fact that could be subject to further study. But it is a negative divergence, and therefore the rule $\{(\mathsf{Stdy.YID{=}I})\} \Rightarrow_{\mathsf{S.SID}} \{(\mathsf{C.Credits{=}10})\}$ would not be generated when considering minimal improvement, since this rule does not improve upon the simpler rule $\{\} \Rightarrow_{\mathsf{S.SID}} \{(\mathsf{C.Credits{=}10})\}$. To also allow significant negative divergences we define **minimal divergence** as follows: all rules $A \Rightarrow C$ must have a confidence c such that for all proper subrules $B \Rightarrow C$ with confidence c' it holds that $|c - c'|$ is greater than a minimal divergence threshold. This pruning measure has the effect of eliminating rules that have additional constraints but a confidence similar to a simpler rule, which we prefer.

Definition 9. *The* ***divergence*** *of an association rule $A \Rightarrow_K C$ is defined as*

$$\max_{B \subset A} |conf(A \Rightarrow_K C) - conf(B \Rightarrow_K C)| \, .$$

6. Experiments

In this section we consider the results of several experiments performed on synthetic and real world databases. The SMuRFIG algorithm[1] was implemented in C++, and run on a system with a 2.16 GHz processor and 2GB RAM.

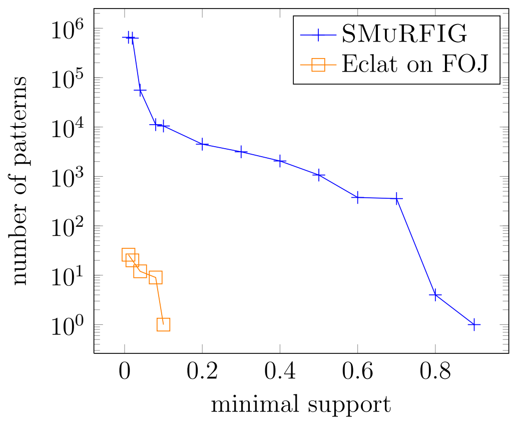
First, we consider a snapshot of the student database from the University of Antwerp's Computer Science department. The scheme roughly corresponds to the one given in Figure 1. There are 174 courses, 154 students and 40 professors, 195 links between professors and courses, and 2949 links between students and courses. The second database comes from the KDD-Cup 2003[2], and comprises a large collection of High Energy Physics (HEP) papers. It consists of HEP papers linked to authors, journals, and also to other papers (citations). It contains 2543 papers, 23621 authors and 76 journals, and there are 5012 connections between authors and papers, plus 458 connections from papers to journals.

6.1. **Patterns.** Several interesting patterns were discovered in the Student database, and we now discuss a few of them. We found $\{(\mathsf{C.Room} = \mathtt{G010})\}_{\mathsf{S.SID}}$ with a support of 81%, representing that 81% of the students take a course given in room `G010`. The itemset $\{(\mathsf{C.Room} = \mathtt{G010}), (\mathsf{P.ID} = \mathtt{DOE})\}_{\mathsf{S.SID}}$ has a support of 76%. Together they form the association rule $\{(\mathsf{C.Room} = \mathtt{G010})\} \Rightarrow_{\mathsf{S.SID}} \{(\mathsf{P.ID} = \mathtt{DOE})\}$ with a confidence of 94%, which means that 94% of the students that take a course in room G010 also take a course taught by professor '`DOE`' in `G010`. We found $\{(\mathsf{S.Study} = \mathtt{1\text{-}MINF\text{-}DB})\}_{\mathsf{P.PID}}$ with a support of 63%, stating that 63% of the professors teach a course that is taken by students of `1-MINF-DB`. We also find this pattern with a different key $\{(\mathsf{S.Study} = \mathtt{1\text{-}MINF\text{-}DB})\}_{\mathsf{S.SID}}$ with a support of 7%, telling us that only 7% of the students belong to `1-MINF-DB`. This is a clear example of the merit of key-based frequency. Another example is the itemset $\{(\mathsf{S.Study} = \mathtt{BINF})\}$ which has 68% PID support, 75% SID support and 39% CID support. Hence, 68% of all professors teach a course taken by students from `BINF`, 75% of the students are in `BINF` and 39% of the courses are taken by `BINF` students.

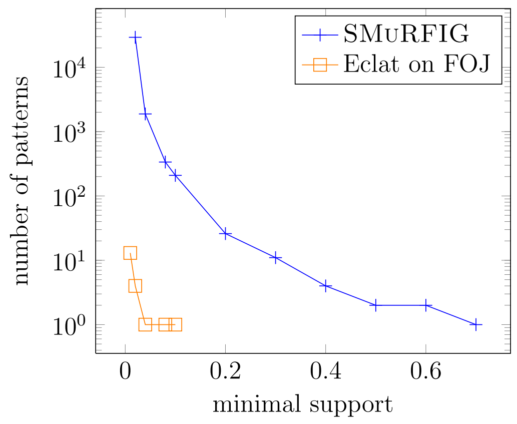
Some patterns found in the HEP database include the following association rules: $\{(\mathit{earliest\_year}{=}\mathsf{2002})\} \Rightarrow_{\text{paper.id}} \{(\mathit{published}{=}\mathtt{false})\}$ and $\{(\mathit{earliest\_year}{=}\mathsf{2003})\} \Rightarrow_{\text{paper.id}} \{(\mathit{published}{=}\mathtt{false})\}$ with respectively 60% and 93% confidence. Since the dataset is from the 2003 KDD-Cup, this tells us that most recently submitted papers ($\mathit{earliest\_year} > \mathsf{2002}$) are not yet published. A rule such as $\{(\mathsf{authors}{=}\mathsf{2})\} \Rightarrow_{\text{paper.id}} \{(\mathsf{published}{=}\mathtt{true})\}$ with 67% confidence versus the rule $\{(\mathsf{authors}{=}\mathsf{4})\} \Rightarrow_{\text{paper.id}} \{(\mathsf{published}{=}\mathtt{true})\}$ with 80% confidence show us that a paper with four authors is more likely to be accepted than a paper with only two authors. The association rule $\{(\mathsf{cited}{=}\mathsf{0})\} \Rightarrow_{\text{paper.id}} \{(\mathsf{published}{=}\mathtt{false})\}$ with 60% confidence, tells us that if a paper is not cited by another paper in the database, in 60% of the cases it is not published. Furthermore, the rule $\{(\mathsf{cited}{=}\mathsf{2})\} \Rightarrow_{\text{paper.id}} \{(\mathsf{published}{=}\mathtt{true})\}$

[1] SMuRFIG can be downloaded at `http://www.adrem.ua.ac.be/`

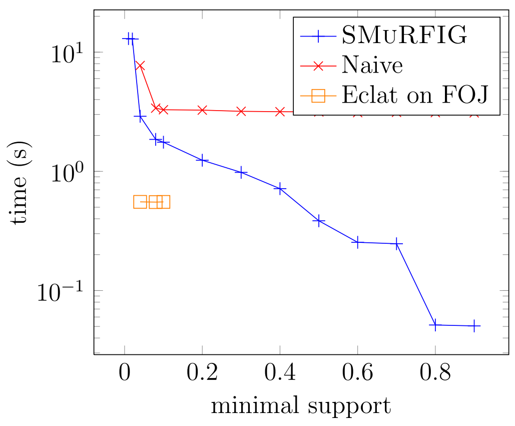
[2] `http://kdl.cs.umass.edu/data/`



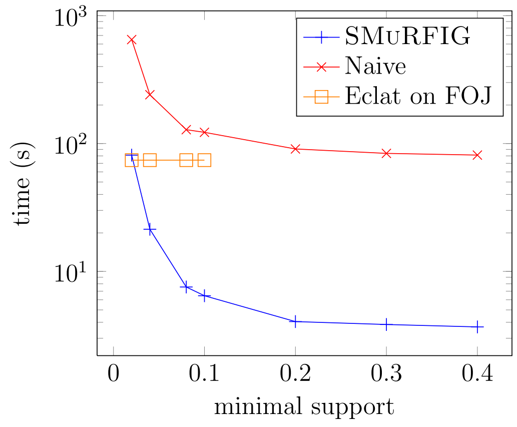
(A) **Student DB:** Number of patterns



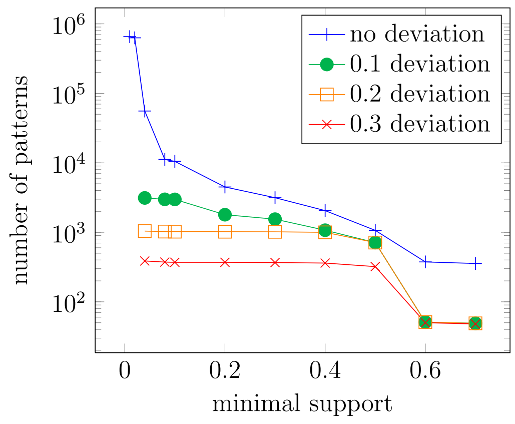
(B) **HEP DB:** Number of patterns



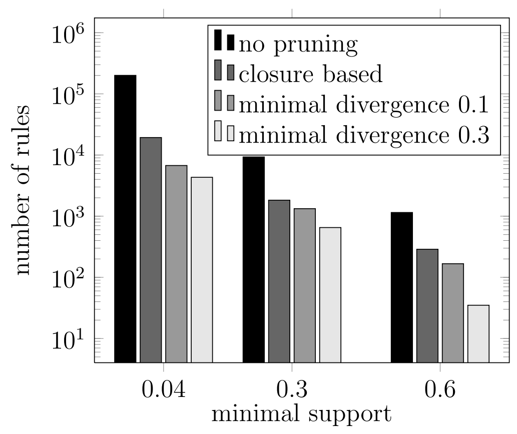
(C) **Student DB:** Runtime



(D) **HEP DB:** Runtime



(E) **Student DB:** Results of pruning itemsets for varying minimal deviation



(F) **Student DB:** Results of pruning rules having a minimal confidence of 0.5

FIGURE 3. Experimental Results for the Student and HEP Databases.

with confidence 65% says that if a paper is cited by two other papers, then in 65% of the cases it is published. Further rules show that this number steadily increases and a paper cited eight times is published in 85% of the cases. Similarly, we found rules expressing that the number of papers cited in a paper also increases its publication probability. We also found the rule $\{(\mathsf{class}{=}\mathtt{Quantum\ Algebra})\} \Rightarrow_{\text{author.id}} \{(\mathsf{published}{=}\mathtt{true})\}$ with 75% confidence, expressing that 75% of the authors who write a paper in `Quantum Algebra` get one published. These examples clearly demonstrate that relational itemsets and association rules with key-based frequency allow us to find interesting, easy to interpret patterns.

To evaluate the deviation measure, we carried out an experiment on the Student DB with varying support and deviation thresholds. As is apparent in Figure 3e, increasing the minimal deviation effectively reduces the number of itemsets. This reduction works as promised, for example, the previously mentioned frequent itemset $\{(\mathsf{C.Room} = \mathtt{G010})\}_{\mathsf{S.SID}}$ with support 81% has a deviation of 22% thus is considered interesting. The itemset $\{(\mathsf{C.Room} = \mathtt{G006})\}_{\mathsf{S.SID}}$ has 53% support, but it only has a deviation of 2%, thus is considered less interesting.

We also ran experiments on the Student dataset to evaluate closure and minimal divergence. The results are shown in Figure 3f. We observe that lossless closure-based redundancy pruning reduces the output by an order of magnitude, and thus works quite well. Using (only) minimal divergence we significantly reduce the rule output of our algorithm by several orders of magnitude. When examining the results we observed that divergence-based pruning indeed eliminates undesired results. For instance, we found the rule $\{\} \Rightarrow_{\mathsf{S.SID}} \{(\mathsf{contracttype}{=}\mathtt{diploma})\}$ with a confidence of 99%. Using this fact, and a divergence threshold of 10% we now prune many rules such as $\{(\mathsf{P.ID}{=}\mathtt{DOE})\} \Rightarrow_{\mathsf{S.SID}} \{(\mathsf{contracttype}{=}\mathtt{diploma})\}$ with different professors, rules such as $\{(\mathsf{C.Room}{=}\mathtt{US103})\} \Rightarrow_{\mathsf{S.SID}} \{(\mathsf{contracttype}{=}\mathtt{diploma})\}$ with different rooms, and so on.

6.2. **Performance.** We experimentally evaluated the performance of the Naive and SMuRFIG algorithms on the Student and HEP datasets, gradually varying the minimum support threshold. In our experiments we also ran the standard categorical Eclat algorithm [4] on the full join table, an approach taken in some previous works. The number of patterns and the runtimes are reported below.

In Figures 3a and 3b we see that the Eclat algorithm applied directly to the join table finds far fewer patterns than SMuRFIG (and Naive which mines the same patterns) on both the Student and HEP databases. Since the (relative) minimum support threshold is set against the size of the full outer join of the database and not the size of an individual table, an itemset must be very connected to have a high support in this join. Clearly, many interesting patterns that are not highly connected will be discarded this way. Apart from this, the support of an itemset in this join table is of course less interpretable.

The runtimes reported in Figures 3c and 3d clearly indicate that the Naive algorithm takes a lot more time than SMuRFIG, often up to an order of magnitude or more. Although SMuRFIG performs more operations per itemset than Naive does, the Naive algorithm operates on an extremely large data table, while SMuRFIG works with the original database, which pays off in terms of runtime. Note that for the HEP database, SMuRFIG is also faster that the Eclat algorithm on the join table, even though the latter finds far fewer patterns.

6.3. **Scalability.** To test the scalability of our algorithm we ran SMuRFIG on a collection of synthetically generated databases, with a varying number of entities, tuples, and attributes per entity. These databases were generated as follows[3]. For a given number of entities, a schema is created such that it has a tree shape. Each entity table has a number of attributes that is randomly chosen from a given interval, and each of these attributes has a random number of possible values, drawn from a given interval. The number of tuples per entity is also uniformly picked from a given interval. The entity tables are generated using a Markov chain, i.e. each attribute is a copy of the previous attribute with some given probability, otherwise its value is chosen uniformly at random. The binary relation tables which connect the entities can be seen as bipartite graphs, which are instantiated with a random density drawn from a given interval. These graphs are generated such that the degree distributions of the columns obey a power law[4]. The exponents of these power laws can be computed directly from the density.

For our experiments we chose the number of attributes per table to lie between 5 and 10, the number of values per attribute between 10 and 20, the number of tuples per entity between 10^3 and 10^4, and the relation density between 0.01% and 0.02%. The minimum support threshold for SMuRFIG was set to 1%.

In Figure 4a we see that the time spent per frequent itemset increases linearly with the number of entities in the database. However, in Figure 4b we see that the time per pattern (i.e. the average time spent for all candidate patterns) *decreases* as the number of entities grows, while the complexity analysis in Section 3 says that it should increase, since more intersections and propagations are performed. This apparent contradiction can be explained by the fact that in our analysis we consider the largest size of a KeyID list for some entity E to be $|E|$, while in reality they are often much smaller. In our experiments the average size of the KeyID lists decreases so fast, that intersecting or propagating them actually takes less time. If we were to implement SMuRFIG with, say, bitvectors of constant size, then we would see an increase in the time per pattern. Figure 4c shows that as we increase the number of tuples per entity, the time required by SMuRFIG increases linearly, which was predicted. Finally, in Figure 4d we see that the number of attributes has no effect on the time per pattern. The number of attributes only increases the number patterns, not the time it takes to compute the support of a pattern.

[3] The Python code used to generate the synthetic relational databases can be downloaded at `http://www.adrem.ua.ac.be`.

[4] We conjecture that many realistic databases obey a power law to some extent, i.e. many tuples have a few connections and a few tuples have many connections.

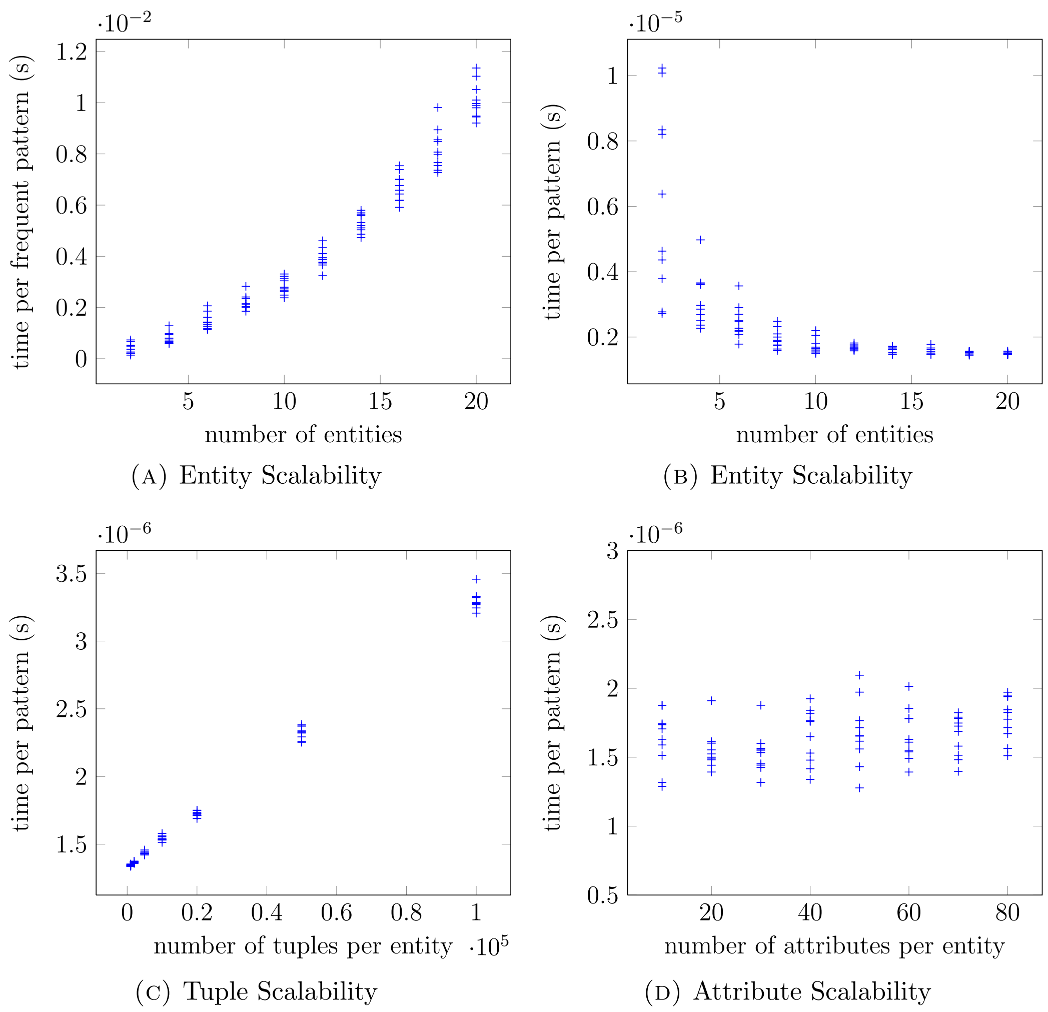


Figure 4. Scalability experiments using synthetically generated relational databases.

7. Related Work

Our work is related to more general frequent query mining approaches such as Conqueror [9] and Warmr [8]. Since the pattern types considered there are more general and complex, and especially since they use different support measures, a direct and fair comparison cannot easily be made. Other general approaches include tree and graph mining [7, 11], which can be seen as data-specific special cases of the general query mining strategy. Often, adapted query mining techniques are therefore used for this purpose. On the other hand, we are not aware of many approaches considering the more specific case of (categorical) itemsets in a multi-relational setting. Koopman and Siebes' R-KRIMP [12] and the algorithm by Crestana-Jensen and Soparkar [5] both use the number of occurrences of the itemset in the join of all tables as the frequency measure. This is done without fully computing or at least storing this join. Support is basically computed using standard techniques; Crestana-Jensen and Soparkar is based on Apriori [2] and R-KRIMP is based on KRIMP [14], both with the addition of some case-specific optimizations. These algorithms return the same results as Apriori (∼KRIMP) run on the full join. This is not the case for our algorithm, since we compute supports simultaneously for multiple keys. Furthermore, compared to the two-phased approach of Crestana-Jensen and Soparkar the computation of intra- and inter-entity itemsets is merged in our algorithm, and is performed in a depth-first manner. Ng et al. [13] focus on star schemes, in which case the join table is essentially already materialized as the fact table. This approach closely mirrors the two-phased approach of Crestana-Jensen and Soparkar, but adds some optimizations specific to star schemes.

None of these relational itemset approaches take into consideration the semantic consequences of the blow-up that occurs in the joining of all tables. If, for example, we have a professor named Susan that teaches a course which is taken by almost all students, in the join of all tables a lot of tuples include the attribute (P.Name=Susan). Thus, $\{(\mathsf{P.Name}{=}\texttt{Susan})\}$ is a frequent relational itemset. Does this mean that a large percentage of professors is named Susan? On the contrary, only one is. All of the mentioned relational itemset mining approaches return the itemset as frequent with respect to the join of all tables, which is not that useful. In our approach only $\{(\mathsf{P.Name} = \texttt{Susan})\}_{\mathsf{S.SID}}$ is a frequent itemset, immediately giving the information of what aspect of this itemset determines its frequency.

Cristofor and Simovici [6] do define a support measure similar to ours called *entity support*. For those itemsets consisting of attributes from one entity, next to the join support, they also consider the support in the number of unique tuples of that entity. In other words, they do consider KeyID-based support, but only for inter-entity itemsets and only in the key of that entity. They are therefore unable to express a pattern like $\{(\mathsf{P.Name} = \texttt{Susan})\}_{\mathsf{S.SID}}$. Furthermore, their proposed Apriori based algorithm is only defined for star schemes, and explicitly computes joins, which as we already stated does not scale well to larger databases.

8. Conclusions and Future Work

In this paper we defined frequent relational itemsets using a novel support measure, based on the keys of the relational database scheme. We implemented an efficient depth-first propagation-based algorithm for mining these frequent relational itemsets. Our experiments showed that the SMuRFIG algorithm performs well on real datasets, is scalable, and is capable of finding interesting patterns. This stands in contrast with most existing work, where the used support measure being used is unintuitive or inefficient to compute. In addition, we defined the deviation measure in order to address the statistical pattern blow-up that is specific to the relational context, and we experimentally showed that it works as promised. Part of our future work is to study this measure further and extend it to inter-entity itemsets. Furthermore, we generalized some popular redundancy measures - closure-based redundancy and minimal improvement - to the multi-relational setting, and confirmed that they can reduce the output when complete results are not desired.

For the sake of clarity, we restricted ourselves to simple schemes in this work. The simple scheme setting already allows us to mine a large collection of databases (or parts of databases), and can result in interesting discoveries, as we have seen in Section 6. Preliminary experiments show that some small extensions to our definitions allow us to find more complex patterns while only suffering a relatively small loss of efficiency. For instance, allowing non-key attributes to occur in relations next to entities adds a great amount of expressive power that can also be found in the general frequent query mining case. It also forms the basis for extending to more general graph-shaped schemes where more than one path may exist between two entities. The study of the algorithmic implications of such extensions is part of our future work. Furthermore, our current approach has no way to deal with cyclic relations. We plan to investigate the possibility of a simple extension where an entity is allowed to occur a fixed number of times. Naively this could be done by just copying the table of the entity and updating the relations accordingly. Unfortunately, duplicate computations and some semantic consequences would arise. Finally, we plan to look into additional quality and ranking measures related to the peculiarities of the relational case, in order to further improve end-user output.

Acknowledgements

Wim Le Page is supported by the Interuniversity Attraction Poles Programme – Belgian State – Belgian Science Policy. Michael Mampaey is supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, volume 22:2 of *SIGMOD Record*, pages 207–216. ACM Press, 1993.

[3] R. Bayardo Jr, R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery*, 4(2):217–240, 2000.

[4] T. Calders, B. Goethals, and M. Mampaey. Mining itemsets in the presence of missing values. In Y. Cho, R. L. Wainwright, H. Haddad, S. Y. Shin, and Y. W. Koo, editors, *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 404–408. ACM, 2007.

[5] V. Crestana-Jensen and N. Soparkar. Frequent itemset counting across multiple tables. In *Proceedings of the 4th Pacific-Asia Conference (PAKDD), Kyoto, Japan, April 18-20, 2000*, pages 49–61, 2000.

[6] L. Cristofor and D. Simovici. Mining association rules in entity-relationship modeled databases. Technical Report 2001-1, University of Massachusetts Boston, 2001.

[7] J. De Knijf. Fat-miner: mining frequent attribute trees. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007*, pages 417–422, 2007.

[8] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

[9] B. Goethals, W. Le Page, and H. Mannila. Mining association rules of simple conjunctive queries. In *Proceedings of the SIAM International Conference on Data Mining (SDM), April 24-26, 2008, Atlanta, Georgia, USA*, pages 96–107, 2008.

[10] B. Goethals, W. L. Page, and M. Mampaey. Mining interesting sets and rules in relational databases. In *Proceedings of the 25th ACM Symposium on Applied Computing*, 2010.

[11] E. Hoekx and J. Van den Bussche. Mining for tree-query associations in a graph. In *Proceedings of the Sixth International Conference on Data Mining (ICDM)*, pages 254–264, Washington, DC, USA, 2006. IEEE Computer Society.

[12] A. Koopman and A. Siebes. Discovering relational item sets efficiently. In *Proceedings of the SIAM International Conference on Data Mining (SDM), April 24-26, 2008, Atlanta, Georgia, USA*, pages 108–119, 2008.

[13] E. Ng, A. Fu, and K. Wang. Mining association rules from stars. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, volume 20, pages 30–39, 2002.

[14] A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In J. Ghosh, D. Lambert, D. B. Skillicorn, and J. Srivastava, editors, *Proceedings of the SIAM International Conference on Data Mining 2006 (SDM)*, pages 393—404. SIAM, 2006.

[15] M. J. Zaki. Mining non-redundant association rules. *Data Mining and Knowledge Discovery*, 9(3):223–248, 2004.

[16] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proceedings of the 3rd Internationl Conference on Knowledge Discovery and Data Mining (KDD)*, pages 283–286, 1997.