Mining Constraint Violations

Stefano Ceri, Francesco Di Giunta, Pier Luca Lanzi
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci 32–20133 Milano, Italy
{ceri,digiunta,lanzi}@elet.polimi.it

In this paper, we introduce *pseudo–constraints*, a novel data mining pattern aimed at identifying rare events in databases. At first, we formally define pseudo–constraints using a probabilistic model and provide a statistical test to identify pseudo–constraints in a database. Then, we focus on a specific class of pseudo–constraints, named *cycle pseudo–constraints*, which often occur in databases. We define cycle pseudo–constraints in the context of the ER model and present an automatic method for detecting cycle pseudo–constraints from a relational database. Finally we present an experiment to show cycle pseudo–constraints "at work" on real data.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*; G.3 [**Probability and Statistics**]: —*contingency table analysis, multivariate statistics*

General Terms: Algorithms, Experimentation, Human Factors, Theory

Additional Key Words and Phrases: Deviation detection, Probabilistic models, Relational data mining

1. INTRODUCTION

Integrity constraints are predicates that must be true on any database state. They are defined by the database designer as a requirement for the database consistency: they hold initially and then they must remain true throughout the life of the database application. When they evaluate to false, the action that causes this event (called a constraint violation) is undone, thus enabling the filtering of those state changes that violate the database consistency [Garcia-Molina et al. 2001].

In order to test the consistency of a database state relative to a given constraint, systems often compute its *denial query*, extracting those tuples that satisfy the constraint predicate in negated form (called the *denial form* of the constraint); consistency holds if the denial query returns an empty result. The tuples extracted by the denial query represent constraint violations; many approaches exist for restoring the database consistency by conditioning the content of these tuples [Ceri and Widom 1990; Widom and Ceri 1995].

In this paper, we are interested in certain predicates, called *pseudo–constraints*, which need not be true on every instance; however, if interpreted as constraints, these predicates have significantly few violations; moreover, due to the peculiar structure of pseudo–constraints, such violations represent "interesting facts". For instance, a constraint could state that "*no engineering graduate student can have an undergraduate degree in human sciences*", but such a constraint does not hold on most engineering universities. However, if we consider the above as a

pseudo–constraint, it probably has very few violations, and these constitute an interesting student population, retrieved by the query that denies the pseudo–constraint.

The purpose of this paper is, in the first place, to introduce pseudo–constraints, i.e., predicates which – once interpreted as constraints – have significantly few and interesting violation instances. We discuss pseudo–constraints from an intuitive point of view, we give their general structure and progressively show which probabilistic properties formally define them. We provide a solid statistical method (whose significance level is discussed) to test if a predicate is actually a pseudo–constraint.

Next, we introduce a broad class of pseudo–constraint called *cycle pseudo–constraint*, based upon cyclic relationships among entities. An example of cycle pseudo–constraint is prescribing that only resident citizens of a given country can represent that country in an international committee: violations would then be the rare citizens who represent a country without being a resident. These are pseudo–constraints of a given structure, and thus we can present a method for automatically detecting them.

It is important to note that, while integrity constraints are designed at compile time, pseudo–constraints are discovered when the database is in operation; therefore, pseudo–constraints properly belong to data mining concepts. However, they are schema-level properties, structurally identical to integrity constraints; once asserted, their violation instances can be extracted several times, by issuing simple queries. Their conceptual nature is such that they can be tentatively defined by database designers and submitted to a verification tool; when they satisfy the pseudo–constraint conditions, then the tool asserts them as pseudo–constraints and extracts the violations. The major difference between pseudo–constraints and association rules is that the latter are not schema-level properties and deal with frequency measures instead of real world probabilities and probabilistic models. Many other differences between pseudo-constraints and other data mining patterns, including association rules, will be clarified in Section 9, which is dedicated to related work.

Our definition of pseudo–constraints is based upon the Entity-Relationship (ER) model. We prefer to explain pseudo–constraints using a conceptual model because this notion is model-independent (and indeed applies as well to object-oriented or XML databases). However, we show how to search pseudo–constraints in a relational setting, for given mappings of ER schema into relational tables. In this way, we show the SQL queries that allow us to decide if a given predicate is a pseudo–constraint.

The general problem of mining all pseudo–constraints, regardless of their format, is intractable, because these could be arbitrarily complex expressions with an unbound number of simple predicates. However, this paper opens up opportunities for defining other classes of pseudo–constraints; once new classes have been structurally defined, it will be possible to build suitable detection methods.

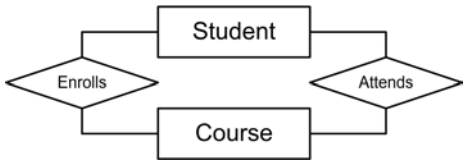

Fig. 1. ER schema of the database of Example 2.1 about students and courses

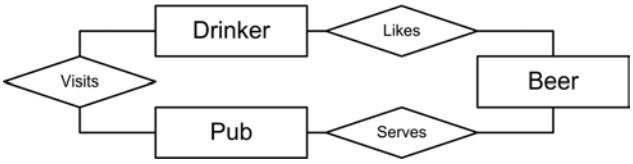

Fig. 2. ER schema of the database of Example 2.2 about drinkers, pubs and beers

2. INFORMAL DEFINITION OF PSEUDO–CONSTRAINTS

In this section, we introduce the intuitive notion of pseudo–constraint by using some simple examples. Their common pattern, described informally as a *generic rule*, will be formalized as a *pseudo–constraint* in the next section.

EXAMPLE 2.1. *Consider a database concerning students and courses, with two relationships linking students to the courses in which they are enrolled and to the courses they attend; the ER schema is shown in Figure 1. A typical constraint requires that students can only attend courses after being enrolled. In addition, we may also expect, as a general rule, that "*most of the students enrolled in a course also attend it*". This not a constraint, because attending courses is not mandatory. However, the rule expresses an expected behavior; violations denote situations when students first enrolled into a course, and then missed to attend it, due either to illnesses, or to dissatisfaction, or other causes; perhaps the enrollment was a bureaucratic error. Anyway, violations denote a small but interesting set of <student-course> pairs.*

Structurally, the above rule involves two entities, STUDENT and COURSE, linked by two different relationships, ENROLLS and ATTENDS. It states that a pair of instances ⟨*student*,*course*⟩ linked by the relationship ENROLLS is likely to be linked also by the relationship ATTENDS.

EXAMPLE 2.2. *Consider the classic example of drinkers, beers and pubs (from Exercise* 4.6 *of [Ullman 1980]). The database, whose ER schema is depicted in Figure 2, contains data about drinkers liking beers, drinkers visiting pubs, and beers served in pubs. From our knowledge of human habits, we expect that "*most drinkers visit pubs that serve beers that they like*". A violation to the above rule would be represented by a drinker who visits a pub which does not serve any beer that the drinker likes.*

The rule described in this example is similar to the rule of Example 2.1. It involves the entities DRINKER and PUB, linked by the relationships VISITS and by

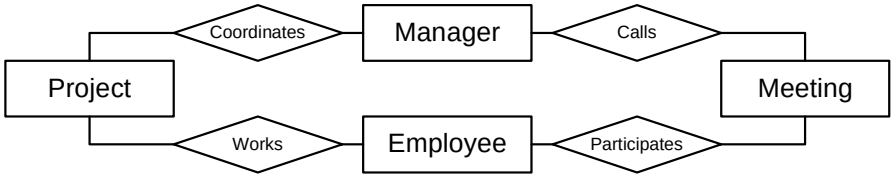

Fig. 3. ER schema of the database of Example 2.3 about managers, employees, projects, and meetings

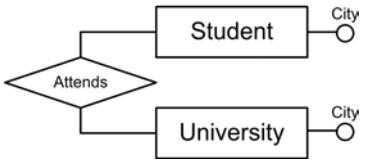

Fig. 4. ER schema of the database of Example 2.4 about students and their universities

the composition of relationships SERVES∘LIKES (we will denote composition with the "∘" notation). The rule states that a pair ⟨*drinker*,*pub*⟩ linked by VISITS is likely to be linked also by the composition SERVES∘LIKES.

EXAMPLE 2.3. *Let us next consider the ER schema in Figure 3. Managers coordinate projects and call meetings, employees work in projects and participate to meetings. Although not forbidden, it would be quite unusual that a manager and an employee involved in the same project never take part to the same meeting. Therefore, we expect that: "*if a manager m and an employee e are connected because m coordinates a project and e works in that project, then there is also at least one meeting called by m and attended by e*". A violation gives evidence to something undesirable (and therefore interesting), for example that a manager does not meet his employees, or that a lazy employee does not take part to the project's meetings.*

Also the rule shown in this example has the pattern that was introduced in Examples 2.1 and 2.2. It states that a pair of entity instances ⟨*manager*,*employee*⟩ linked by the relationship composition COORDINATES∘WORKS is likely to be linked also by CALLS∘PARTICIPATES.

EXAMPLE 2.4. *Finally, consider a database involving students and the universities they attend. The ER schema is depicted in Figure 3. From common sense knowledge we expect that "*if a student attends a university, then the student lives in the city where the university is located*". Violations include all student–university pairs with a city mismatch, representing the "non–resident" students.*

As in the previous cases, Example 2.4 refers to two entities (STUDENT and UNIVERSITY) and two links between them: the first link is a relationship (ATTENDANCE), while the second link is built by pairing instances of *student* and *university* whose City attributes have equal values. The rule states that when a pair ⟨*student*,*university*⟩ is linked by the relationship ATTENDS, the same entities

normally have the same City value.

A somehow different example is the following.

EXAMPLE 2.5. *Assume that the entity* STUDENT *of the previous example has, in addition to* City*, also attributes* BirthDate *and* Level*, with* Level *taking values undergraduate and graduate. A possible rule is: "*graduate students are over 14 years old*". This is not a constraint, but violation instances constitute a tiny and fairly interesting population of gifted students.*

The structure of this last rule differs from those of previous examples because no relationships are used; however, the rule still involves two predicates (being graduate student, having a certain age) such that the instances satisfying the first one should normally satisfy the second one.

All the rules in Examples 2.1 to 2.5 share the following features:

—The rule applies to a specific *population* (or "domain") extracted from the database. The population is a pair of entities in Examples 2.1 to 2.4 and a single entity in Example 2.5.

—Two predicates P_1 and P_2 are stated on the rule's population. Every predicate is a formula which can be computed for any given element of the population, yielding *true* or *false*.

—The rule states that when an element of the population satisfies P_1, it *usually* satisfies also P_2.

—A *rule violation* is an element of the population which satisfies P_1 but not P_2.

We are now ready to give an informal definition of pseudo–constraint:

Def 2.6 *Pseudo–constraint.* Given a domain D, a *pseudo–constraint* PC is a pair $\langle P_1, P_2 \rangle$, where P_1 and P_2 are predicates over D, such that if P_1 holds then *usually* also P_2 holds and therefore there are *few* rule violations. We denote a pseudo–constraint as:

$$PC: \qquad P_1 \rightsquigarrow P_2$$

The exact definition of the meaning of the words "usually" and "few" will be given in the next section. Note that pseudo–constraints are directional, therefore $P_1 \rightsquigarrow P_2$ does not imply $P_2 \rightsquigarrow P_1$.

3. FORMAL DEFINITION OF PSEUDO–CONSTRAINTS

In this section, we identify the relevant parameters for defining pseudo–constraint, conveniently organized them in a *contingency table*. Then, we build the *probabilistic model* for these parameters; this constitutes the *formal definition* of the notion of pseudo–constraint.

3.1 Relevant parameters and contingency tables

According to the informal definition in Section 2, a pseudo–constraint $PC: P_1 \rightsquigarrow P_2$ is a couple of predicates $\langle P_1, P_2 \rangle$ on the same domain D, such that P_1 and P_2 have some peculiar traits. It is therefore clear that the relevant parameters in a database instance are:

	P_1	$\overline{P_1}$	
P_2	$n_{1,1}$	$n_{1,2}$	$n_{1,\cdot}$
$\overline{P_2}$	$n_{2,1}$	$n_{2,2}$	$n_{2,\cdot}$
	$n_{\cdot,1}$	$n_{\cdot,2}$	n

Fig. 5. Count contingency table for a candidate pseudo–constraint $PC: P_1 \rightsquigarrow P_2$; $n_{i,\cdot}$ and $n_{\cdot,j}$ represent partial totals; n is the total number of instances of the population.

	P_1	$\overline{P_1}$	
P_2	$p_{1,1}$	$p_{1,2}$	$p_{1,\cdot}$
$\overline{P_2}$	$p_{2,1}$	$p_{2,2}$	$p_{2,\cdot}$
	$p_{\cdot,1}$	$p_{\cdot,2}$	1

Fig. 6. Probability contingency table for a candidate pseudo–constraint $PC: P_1 \rightsquigarrow P_2$. Note that $p_{2,2} = 1 - p_{1,1} - p_{1,2} - p_{2,1}$.

—$n_{1,1}$ number of instances of D that satisfy both P_1 and P_2;
—$n_{1,2}$ number of instances of D that satisfy P_2 but not P_1;
—$n_{2,1}$ number of instances of D that satisfy P_1 but not P_2;
—$n_{2,2}$ number of instances of D that satisfy neither P_1 nor P_2.

This information can be collected within a *count contingency table* associated to D, as in Figure 5.

In order for the candidate to be a real pseudo–constraint the numbers in its count contingency table should show some suitable traits. But a count contingency table refers to a particular database state, while we want the notion of pseudo–constraint to be an assertion on the underlying domain which is only occasionally represented by a given database state. Therefore, we adopt a probabilistic point of view: we will not use the $n_{i,j}s$ to define pseudo–constraints but rather their underlying probabilities, which we will denote as $p_{i,j}s$ (for example, $p_{1,1}$ is the "real world" probability that an element of D satisfies both P_1 and P_2). This amounts to saying that a count contingency table is an observation of a four–dimensional multinomial random vector (see e.g., [Kendall and Stuart 1963]) whose parameters are n and $\boldsymbol{p} = (p_{1,1}, p_{1,2}, p_{2,1})$, and that can be concisely described in a contingency table associated to PC, named the *probability contingency table*, as in Figure 6. A model for pseudo–constraints will be an assertion on these probabilities $p_{i,j}$s.

3.2 Probabilistic model

We recall that, according to the informal definition in Section 2, a candidate pseudo–constraint $\langle P_1, P_2 \rangle$ is a pseudo–constraint $PC: P_1 \rightsquigarrow P_2$ if the following holds:

(1) if predicate P_1 holds then *usually* also P_2 holds;
(2) violation instances (satisfying P_1 but not P_2) are *few*.

We can rewrite the above assertions using probabilities:

(1) the probability of P_2 given P_1 should be very high;

(2) the probability of P_2 given P_1 should be much greater than the probability of P_2 itself.

In symbols:

$$\mathbb{P}[P_2|P_1] \simeq 1 \tag{1}$$

and

$$\mathbb{P}[P_2|P_1] \gg \mathbb{P}[P_2] \tag{2}$$

These formulas can be "quantified" by means of user–dependent parameters, whose setting leads to different strengths of the notion of pseudo–constraint. A simple way to achieve this result for both Formula (1) and (2) is to state that $\mathbb{P}[P_2|P_1]$ should lie in a suitable portion of the interval $(\mathbb{P}[P_2], 1]$:

$$\mathbb{P}[P_2|P_1] > \mathbb{P}[P_2] + \rho \cdot (1 - \mathbb{P}[P_2]) \tag{3}$$

where ρ is a user–dependent parameter in $(0, 1)$ whose value should be close to 1. We have chosen $\rho = 0.8$ as a default value; rising the value of ρ causes the discovery of fewer pseudo–constraints.

It is easy to see that Inequality (3) implies $\mathbb{P}[P_2|P_1] > \rho$, and, as ρ is close to 1, Formula (1) is satisfied. Furthermore, $\mathbb{P}[P_2|P_1]$ lies in the interval $(\mathbb{P}[P_2], 1]$ much closer to 1 than to $\mathbb{P}[P_2]$, and it is much greater than $\mathbb{P}[P_2]$. Therefore, Formula 2 is satisfied as well.

By expressing the model (3) in terms of the $p_{i,j}$s parameters of the probability contingency table and by rearranging terms, we obtain:

$$\frac{p_{11}}{p_{11} + p_{21}} - \rho - (1 - \rho) \cdot (p_{11} + p_{12}) > 0 \tag{4}$$

If we next define:

$$d(\boldsymbol{p}) = d(p_{1,1}, p_{1,2}, p_{2,1}) = \frac{p_{1,1}}{p_{1,1} + p_{2,1}} - \rho - (1 - \rho) \cdot (p_{1,1} + p_{1,2}) \tag{5}$$

we can then express our model (4) in the following way:

$$d(\boldsymbol{p}) > 0 \tag{6}$$

We can finally give the formal definition of pseudo–constraints:

Def 3.1 *Pseudo–constraint.* A candidate pseudo–constraint $PC = \langle P_1, P_2 \rangle$ over a database D is a *pseudo–constraint* if its probabilities $\boldsymbol{p}$ satisfy inequality (6).

Pseudo-constraints do not assert a property of a particular database state, but rather they assert a semantic property of the database schema. Therefore, we need a *statistical test* to check if a candidate pseudo–constraint denotes a semantic property. The test should compare the *null hypothesis* H_0 that the pseudo–constraint does not hold versus the *alternative hypothesis* H_A that it does hold:

$H_0: \quad d(\boldsymbol{p}) \leq 0$
$H_A: \quad d(\boldsymbol{p}) > 0$

The test must have a certain fixed *significance level* α^*, such that, if we find the hypothesis H_0 true, then the probability of erroneously rejecting H_0 should be at most α^*; a typical value is $\alpha^* = 0.05$.

Our hypotheses H_0 and H_A are both composite (i.e., many different values of the parameters satisfy the null hypothesis and many different values satisfy the alternative); moreover, they are "difficult" hypotheses in that they state non–standard relations among the parameters. Finding a reasonable test with a fixed significance level α^* is thus far from trivial. In particular, exact methods (i.e., with a significance level α which is exactly the desired α^*) are computationally unfeasible. We have therefore defined an approximate test (i.e., with a significance level α which approximates the desired level α^*), which is valid for large n, where n is the number of instances in the count contingency table. Here is a summary of the method that we developed to define the test:

(1) We found $\widehat{D}_n$, the *maximum likelihood estimator* of d (based on the count contingency table) and the asymptotic probability distribution of $\widehat{D}_n$.
(2) We used the asymptotic distribution of $\widehat{D}_n$ to find an approximate *confidence interval* of *level* $\gamma = 1 - \alpha^*$ for d.
(3) We used the approximate confidence interval for d to find an approximate test of level α^* for $d \leq 0$.

This progressive derivation is lengthy and is postponed to the Appendix A (see Formula 9).

4. CYCLE PSEUDO–CONSTRAINTS IN THE ER MODEL

We now turn our attention to the class of *cycle pseudo–constraints*, which includes Examples 2.1 to 2.4 of Section 2. Empirically, we discovered that the class includes several relevant pseudo–constraints; moreover, the regular structure of these pseudo–constraints makes their fully automatic detection possible. In this section we focus on the definition of the class on the ER model; in the next Section we consider their translation into the relational model and describe the SQL queries required for computing the count contingency table of a candidate cycle pseudo–constraint; and finally, in Section 6 we describe an efficient method for mining all the cycle pseudo–constraints for a given database.

Cycle pseudo–constraints defined over an ER schema have the following common features:

—their domain is the Cartesian product of two entities (possibly identical) of the ER model of the database;
—the predicates P_1 and P_2 express that two pairs are "linked" in the schema, either directly or by means of link compositions; a link between a pair of entities exists when the two entities are connected by a relationship or when they have the same attribute values.

Note that the composition of the two "links" of any pseudo–constraint form a cycle on the ER schema. Conversely, given any cycle in the ER schema, and considering any two pairs of entities breaking that cycle into two links, we can informally define a cycle pseudo–constraint as the constraint stating that "*whenever a pair of entity instances are connected by one of the links, they are usually also connected by the other one*" (see Figure 7). In addition, for every *positive* form of a cycle pseudo–constraint, there are three *negated forms*, stated as follows:

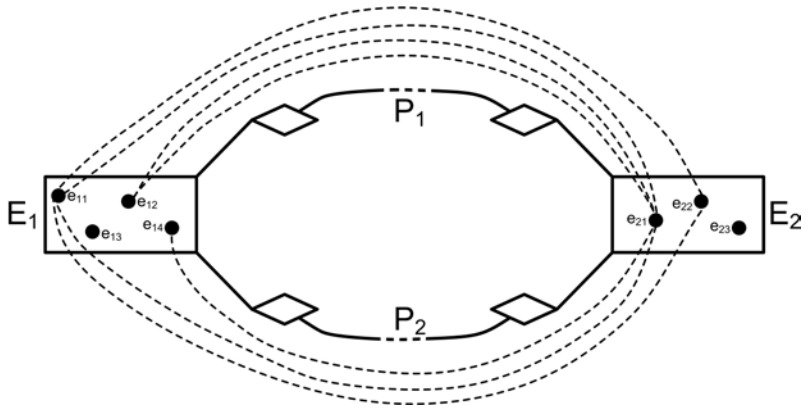

Fig. 7. A cycle pseudo–constraint $P_1 \rightsquigarrow P_2$. E_1 and E_2 are the entities domain of the pseudo–constraint; dotted lines join the instances satisfying P_1 and P_2; $\langle e_{12}, e_{21} \rangle$ is a violation instance

—"*whenever a pair of entity instances are connected by one of the links, they are usually* not *connected by the other one*"

—"*whenever a pair of entity instances are* not *connected by one of the links, they are usually connected by the other one*"

—"*whenever a pair of entity instances are* not *connected by one of the links, they are usually* not *connected by the other one*"

Although negated forms are clearly weaker and less semantically relevant, their contingency tables can be computed from the contingency table of the positive form by matrix transposition and/or row and column exchange, thus checking the existence of negative forms is easy.

Def 4.1. Given an ER schema with attributes A_1 of entity E_1 and attribute A_2 of entity E_2 such that A_1 and A_2 have the same domain, we call *domain relationship* SAME_A_1_A_2 between E_1 and E_2 the relationship that links instances e_1 of E_1 to instances e_2 of E_2 having the same value of A_1 and A_2 respectively:

$$e_1 \text{ SAME\_}A_1\text{\_}A_2 \ e_2 = \begin{cases} \mathsf{true} & \text{if } e_1.A_1 = e_2.A_2 \\ \mathsf{false} & \text{otherwise} \end{cases}$$

We simply use the name SAME_A_1 when the attribute names A_1 and A_2 are identical.

From now on, we will not distinguish between "regular" relationships and domain relationships. We then denote a *link* between two entities E_1 and E_2 as any relationship or composition of relationships of arbitrary length connecting E_1 and E_2, and a *predicate link* on every pair $< e_1, e_2 >$ of entity instances of E_1 and E_2 yielding a true value when e_1 and e_2 are connected by the link, and false otherwise. Finally, we define cycle pseudo–constraint simply as a pseudo–constraint built by means of predicate links:

Def 4.2 *Cycle pseudo–constraint.* A pseudo–constraint $PC : P_1 \rightsquigarrow P_2$ on a database D is a*cycle pseudo–constraint* if P_1 and P_2 are predicate links over two

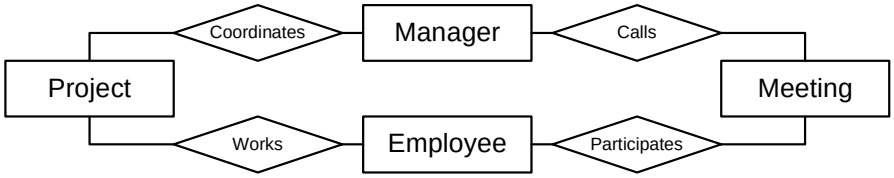

Fig. 8. ER schema of the database of about managers, employees, projects and meetings

entities E_1 and E_2 of D.

We denote cycle pseudo–constraints by a couple of link names, separated by the $\rightsquigarrow$ symbol; for readability we add to link names also the names of the source and target entities. Thus, the pseudo–constraints of Examples 2.1 to 2.4 are named as follows:

(1) Student Enrolls Course $\rightsquigarrow$ Student Attends Course

(2) Drinker Visits Pub $\rightsquigarrow$ Drinker Likes∘Serves Pub

(3) $\begin{cases} \text{Manager Coordinates} \circ \text{Works Employee} \\ \rightsquigarrow \\ \text{Manager Calls} \circ \text{Participates Employee} \end{cases}$

(4) Student Attends University $\rightsquigarrow$ Student Same_City University

5. CYCLE PSEUDO–CONSTRAINTS IN RELATIONAL DATABASES

We now map cycle pseudo–constraints defined in the context of the ER model into their equivalent notion in the context of the relational model. We will focus on Example 2.3 of Section 2; the generalization of the example illustrated in this section is straightforward.

5.1 Mapping ER to relational

In the ER schema of Example 2.3 (which we show again in Figure 8 for the reader's convenience) we identify the following candidate cycle pseudo–constraint:

$$\begin{array}{c} \text{Manager Coordinates} \circ \text{Works Employee} \\ \rightsquigarrow \\ \text{Manager Calls} \circ \text{Participates Employee} \end{array} \tag{7}$$

A standard relational mapping transforms the ER schema of Figure 8 into the relational schema of Figure 9. Note that every cycle in an ER schema is mapped into a cyclic path of foreign keys constraints in the corresponding relational schema; target entities are mapped to target tables.

5.2 Performing the test

Given a candidate pseudo–constraint PC, the test we proposed (Formula (9) in Appendix A) requires the computing of the values $(n_{1,1}, n_{1,2}, n_{2,1}, n)$ of the count contingency table of PC. Our next issue is, therefore, to show how to compute these numbers from a database instance. First of all we note that the 9 numbers in a count contingency table (refer to Figure 5) are not independent, as:

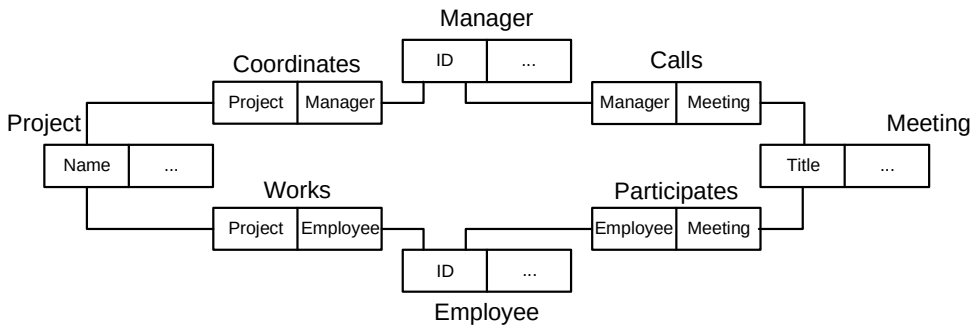

Fig. 9. Relational schema from the ER schema of Figure 8; arcs represent foreign keys

$$\begin{cases} n_{1,1} + n_{1,2} = n_{1,\cdot} \\ n_{2,1} + n_{2,2} = n_{2,\cdot} \\ n_{1,1} + n_{2,1} = n_{\cdot,1} \\ n_{2,1} + n_{2,2} = n_{\cdot,2} \\ n_{1,1} + n_{1,2} + n_{2,1} + n_{2,2} = n \end{cases}$$

So we can compute four independent values (say n, $n_{\cdot,1}$, $n_{1,\cdot}$ and $n_{1,1}$) and then easily obtain the others. This computation can be performed by simple SQL queries. We first compute the two views collecting pairs <employee,manager> such that the employee works for the manager and participates to some meetings called by the manager. We next define their intersection.

SameProject.

```
create view SameProject(ID_Manager, ID_Employee) as
select distinct Manager.ID, Employee.ID
from Manager, Coordinates, Project, Works, Employee
where Manager.ID = Coordinates.Manager and
      Coordinates.Project = Project.Name and
      Project.Name = Works.Project and
      Works.Employee = Employee.ID
```

SameMeeting.

```
create view SameMeeting(ID_Manager, ID_Employee) as
select distinct Manager.ID, Employee.ID
from Manager, Calls, Meeting, Participates, Employee
where Manager.ID = Calls.Manager and
      Calls.Meeting = Meeting.Title and
      Meeting.Title = Participates.Meeting and
      Participates.Employee = Employee.ID
```

SameProjectSameMeeting.

```
create view SameProjectSameMeeting(ID_Manager, ID_Employee) as
```

```
(select *
 from SameProject
     intersect
 select *
 from SameMeeting)
```

The numbers of the count contingency table are computed by counting the number of tuples in such views:

$n_{.,1}$.

```
select count(*)
from SameProject
```

$n_{1,.}$.

```
select count(*)
from SameMeeting
```

$n_{1,1}$.

```
select count(*)
from SameProjectSameMeeting
```

To compute n we just multiply the cardinality of *Manager* (`select count(*) from Manager`) by the cardinality of *Employee* (`select count(*) from Employee`).

Once the contingency table is computed, it is sufficient to use Test (9) of the Appendix to find out if the candidate (7) is indeed a valid pseudo–constraint, with a given significance level α^*.

This example shows how to compute the contingency table of a generic candidate cycle pseudo–constraint. Domain relationships are encoded within the view definitions by omitting a join condition on foreign keys and adding instead a join condition on attribute values.

5.3 Retrieving the violation instances

Violation instances are obtained by materializing the differences of the two views defined earlier, as follows:

SameProject$\overline{\textit{SameMeeting}}$*:*.

```
create view SameProject¬SameMeeting(ID_Manager, ID_Employee) as
 (select *
  from SameProject
      except
  select *
  from SameMeeting)
```

6. MINING CYCLE PSEUDO–CONSTRAINTS

The search of pseudo–constraints requires checking whether a given candidate pseudo–constraint satisfies the statistical test (9). The number of candidates is generally infinite and can be bound only by adopting structural limitations on candidates. Therefore, the search of pseudo–constraints is a difficult problem, and indeed the heuristic method presented in this section applies to cycle pseudo–constraints of bound length, and shows how to reuse results required for the computation of cycles of length $n-1$ in order to test for cycles of length n. Conversely, once a given pseudo–constraint is asserted as a schema property, its violation instances can be retrieved by means of a single query. Such query can be periodically issued to determine new violation instances, i.e., new interesting facts. This query will be repeated by users according to needs, and it is not subject to optimization.

6.1 Finite problem formulation

The cycle pseudo–constraints of all the examples given so far were found in *simple cycles*; i.e., cycles in which every relationship of the ER schema is used at most once in the links. But the definition of cycle pseudo–constraints in Section 4 is not limited to simple cycles; indeed, some interesting pseudo–constraints are found in *arbitrary cycles* which use the same relationships more than once. Given that arbitrary cycles in an ER schema are infinite in number, it is impossible to mine all the cycle pseudo–constraints of a given schema: the mining process has to be limited someway. We have chosen to set a user–defined maximum length l of the cycles.

The detection of all the candidates in all the cycles whose length is at most l is an easy graph–theoretical task. The difficult task is the execution of the queries needed to compute the link predicates so as to fill the contingency tables for the candidates. Link predicates assert that given pairs of tuples of the target tables are connected, and therefore require joining those tables along the paths of foreign key relationships which connect them (see Figure 9); the result of a link predicate computation is simply a set of pairs of identifiers of the target tables.

On the computation of the link predicates, the following easy observation holds: if the link predicates are computed in a suitable order (e.g., growing link length order), then each link predicate can be computed by exactly one join. For instance, if the link predicate relative to L_1 and L_2 has been already computed, the link predicate relative to L_1L_2 can be computed by joining them and projecting over the first and last attribute. Therefore, we have designed an algorithm for reusing the results of queries, whose metrics is the minimization of the number of computed joins. Clearly, employing less than one join for a new link predicate is impossible. The strategy requiring "exactly one join for every link predicate" is, therefore, the best possible strategy according to our metrics. Such computational strategy is defined in the next section; efficient memory management strategies are next discussed.

6.2 Basic algorithm

The basic algorithm to mine cycle pseudo–constraints is the following. Assume that the set of pairs corresponding to links that have already been computed is

kept in memory. At each step, it is known which links are immediately computable by exactly one join from the already computed links and the database tables. The algorithm is then:

(1) choose a link L among the immediately computable links, compute it, and materialize its pairs;
(2) for every already computed link L' between the same target entities as L, compute the contingency tables of the candidates $L \rightsquigarrow L'$ and $L' \rightsquigarrow L$ and their negated forms;
(3) apply the statistical test to these contingency tables and output the possibly found cycle pseudo–constraints;
(4) if there is any other link to compute, go to point (1).

6.3 Algorithm improvements: caching and heuristics

The basic algorithm stores in mass memory link materializations in the format of pairs of tuple identifiers. It can benefit from a suitable caching policy, consisting of storing the same pairs in main memory buffers. Among possible caching schemes, we simply use the *most recently used* policy: links are kept in memory based on their recent use, a property which logically corresponds to favoring the reuse of recent explorations. Mass memory materialization becomes required when main memory is full.

Once decided for a caching of policy, we further improve the algorithm refining the order of computation of the links. In point (1) of the basic algorithm, we can choose the link L to be next computed as the one with the *smallest estimated memory occupancy*; in this way, we heuristically compute as many links as possible (and therefore output as many pseudo–constraints as possible) before the main memory occupancy limit has been reached.

The estimate we use for the memory occupancy of a link is simply a join size estimate; for $T_1 \bowtie_{a_1=a_2} T_2$, such number is $\frac{d_1 \cdot d_2}{d}$, where d_1 is the number of tuples of table T_1, d_2 is the number of tuples of table T_2, and d is the cardinality of the domain of attributes a_1 and a_2. We assume that all domain cardinalities available prior to starting the algorithm, and we substitute at each step the actual size to the estimate size, so that the estimate required at each step is concerned with a single join and not a long chain of joins. This method does not make any assumption on additional optimizations (e.g., indexing, page caching, etc.) which are left to the data management system which actually performs the computation (in our case, Oracle, as we will discuss in the next section). So, the method operates above the DBMS optimizer and is only concerned with the choice of the next query.

7. EXPERIMENTAL RESULTS

To assess the effectiveness of our method, we have developed a tool and conducted several experiments. We first report general applicability requirements, then we extensively report an example, and finally we give some figures of the tool performance.

7.1 Requirements for pseudo–constraints discovery

Based on several experiences, we realized that the method is best applicable when certain requirements hold:

Schema richness. Cycle pseudo–constraints use information about semantically rich databases; the richer is the schema, the greater the likeliness of finding interesting facts.

Data richness. The significance of our statistical analysis requires the use of samples whose dimension should be large enough to justify the approximation of its distribution to its limit–distribution.

Data completeness and correctness. A biased partial data–entry is most harmful for the detection of pseudoconstraints, while the problem is less hard with a random distribution of missing and/or wrong data, as the contingency tables will still be based on random samples from the population of the application domain.

A practical problem is caused by *null values*: should two null values in the same attribute A of two tuples t_1 and t_2 be considered equal? After experiments, we decided to exclude from the computation of the count contingency table of a candidate pseudo–constraint all the tuples with null values in the fields used for testing. In the hypothesis of randomly distributed null values, this exclusion has no effect on the detection of pseudo–constraints.

7.2 Features of the selected database

We report the results of our experiments with anonymous financial data about the accounts of the branches of a Czech bank and their transactions in the years 1993–1998. We have chosen this example because data are public: the database has already been used as a benchmark in the 'Third European Conference on Principles and Practice of Knowledge Discovery in Databases' held in Prague in 1999 [Żytkow and Rauch 1999] and is freely available at the conference web site (http://lisp.vse.cz/pkdd99/) as of the date of our submission.

The Czech bank database fits our requirements since it has a rich relational structure, it has a reasonable size (5369 clients, 4500 accounts, 1056321 transactions), no particular recognizable data entry biases and no evident data entry errors; moreover, the nature of the application domain is such that there can be suspicious anomalies.

The relational structure of the database is depicted in Figure 10. A description of the main tables and their less selfexplaining attributes follows.

Client. Stores information about bank clients:
  birth number. Identifies both the clients birthday and sex.
  district_id. Denotes the district where the client lives.

Account. Stores information about accounts created in all the bank branches:
  district_id. Identifies the district where the account has been created.
  date. Denotes the creation date of the account.
  frequency. Denotes the frequency of statements on the account.

Disposition. Relates clients to their accounts or credit cards:
  type. Can be "owner" or "user".

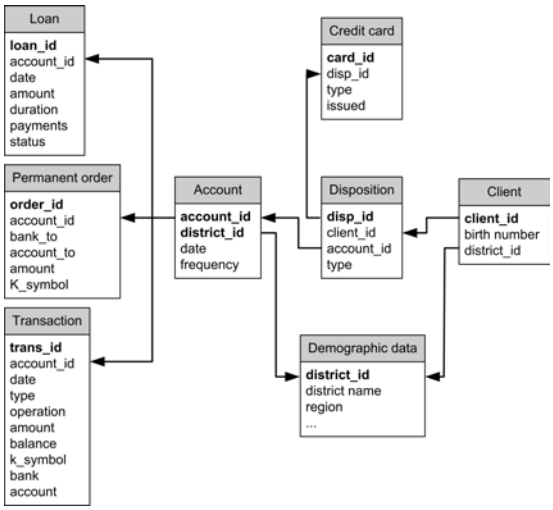

Fig. 10. Relational structure of the Czech bank database

Loan. Describes loans issued by the bank on its accounts.

Permanent order. Describes permanent orders issued by account owners:
bank_to. Denotes the bank of the recipient.
account_to. Denotes the account of the recipient.
K_symbol. Can be "Insurance payment", "Household payment", or "Loan payment".

Transaction. Describes every transaction issued by the clients to third parties:
type. Can be "Credit" or "Withdrawal".
bank. Denotes the bank of the third part.
account. Denotes the account of the third part.

Demographic data. Describes demographic data on the bank districts.

In the experiments we have set parameter $\rho = 0.8$ as default (see Subsection 3.2); tests have been conducted with a significance level $\alpha^* = 0.05$.

7.3 Discussion about discovered pseudo-constraints

Two very similar cycle pseudo–constraints are the following:

PC_Bank_1. If a client has a disposition on an account, the client and the account are in the same district.

PC_Bank_2. If a client has a disposition on an account, the client and the account are in the same region[1].

Details about these pseudo–constraints are in Figure 11.

The violations are clients who have opened accounts far from where they live, in another district or even in another region. This is interesting information that

[1]Note that pseudo–constraint PC_Bank_1 does not imply PC_Bank_2.

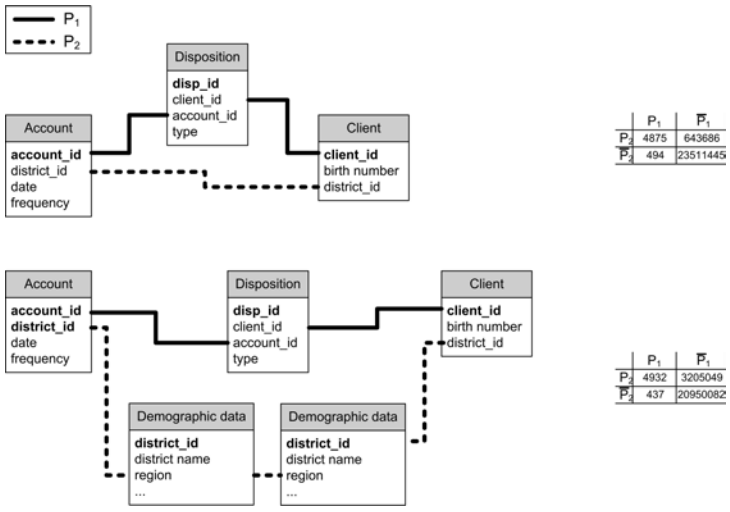

Fig. 11. Structure and contingency tables of pseudo–constraints PC_Bank$_1$:"*if a client has a disposition on an account, the client and the account are in the same district*" ($d(\widehat{\boldsymbol{p}}) = 0.103$) and PC_Bank$_2$:"*if a client has a disposition on an account, the client and the account are in the same region*" ($d(\widehat{\boldsymbol{p}}) = 0.092$).

can be explained in many ways: perhaps the client moved to another district since the date when he created the account, or lives in a district and works in a different one and holds an account in the district where he works, or perhaps some data are erroneous (which is unlikely but not impossible) or someone did not tell the truth about his residence. The bank could investigate about this information. For example, a glance at the violation instances shows that *all the accounts of violation instances have been created in* 1993, which is the first year considered in the bank history, while the creation years of the other accounts range quite uniformly from 1993 to 1997. This suggests that after year 1993 the bank has adopted a different data acquisition and verification protocol or changed something in its organization; the bank managers should be able to provide an interpretation. Anyway, the pseudo–constraint approach has led to the detection of something strange in the data.

Three interesting non cycle pseudo–constraints are the following:

PC_Bank$_3$. Clients with loans usually do not have a credit card.

PC_Bank$_4$. Accounts with loans usually have a permanent order to pay the loan.

PC_Bank$_5$. Accounts with permanent order to pay a loan usually have a loan issued by the bank.

Pseudo–constraint PC_Bank$_3$ is quite unexpected: it is *not* anticipated that people who issued a loan usually do not have a credit card; nevertheless, this happened in the domain of this database, which reports about a slice of the Czech economic reality during years 1993-97 (perhaps, at that time, a person with a credit card was usually well off while a person who asked for a loan was not).

Pseudo–constraint PC_Bank$_4$, instead, seems quite natural: a person, who has to pay a loan, issues a permanent order on its account in order to pay it. Indeed there are no violations to this pseudo–constraint and perhaps we have detected, instead of a pseudo–constraint, a real *constraint* in the bank policy (although not asserted in the database schema).

Pseudo–constraint PC_Bank$_5$ is specular to PC_Bank$_4$ and is equally natural: a person with a permanent order to pay a loan must have a loan to pay, and the loan is usually issued by the same bank. This pseudo–constraint, however, has violation instances: 35 accounts have a permanent order to pay a loan not issued by the bank. This analysis could be used by the bank management in many ways; for example, proposing to the owners of those 35 accounts an internal loan instead of their external one; or instead to restrict the granting of loans to clients who already have an external loan.

Many other pseudo–constraint, both cycle and not, have been found in the database. For example: "two clients with a disposition on the same account are of different gender" which suggests that shared bank accounts are shared by wife and husband (and not, for example, by father and son); "permanent orders issued on the same account have different k_symbol", which suggests that few people issues more than one permanent order for the same purpose (interestingly all the violation instances are couples of permanent orders for household payment).

We end with a couple of pseudo-constraints which we found rather strange:

PC_Bank$_6$. Two (distinct) accounts linked by some transactions to the same third party account have been created on the same date.

PC_Bank$_7$. Two (distinct) accounts linked by some transactions to the same third party account are held by clients with the same gender and birth date.

Details are in Figure 12. Clearly these pseudo–constraints have, at a first glance, no reasonable explanation: why two (different) people who have in common a transaction linking them to the same third party should have the same birth date, sex, and opening date for their accounts? The suspect is that those two people, despite being recorded as two different clients, are actually the same person: the bank has decided (for some reasons) to record a client as two different clients and to provide him with two accounts; some rows of this strange list of accounts and account owners are in Figure 13.

This conjecture is strengthened by looking at the account_id of couples of accounts linked by some transactions to the same third party account: their difference is always about 7400. It is therefore clear that the bank provides "special clients" with two identities and two accounts, one easily identifiable from the other one by means of the 7400–difference rule. The pseudo–constraint approach has thus unexpectedly detected a hidden bank policy.

7.4 Performances

The implementation tool used to carry out the initial experiments discussed in this paper is based on an Oracle server, hosted on a Pentium 4 machine (3Ghz). The tool supports both the general mining of cycle pseudo–constraints with the method of Section 6 (and a given maximum length) and user–defined mining of arbitrary pseudoconstraints; it supports the selection of violation instances even

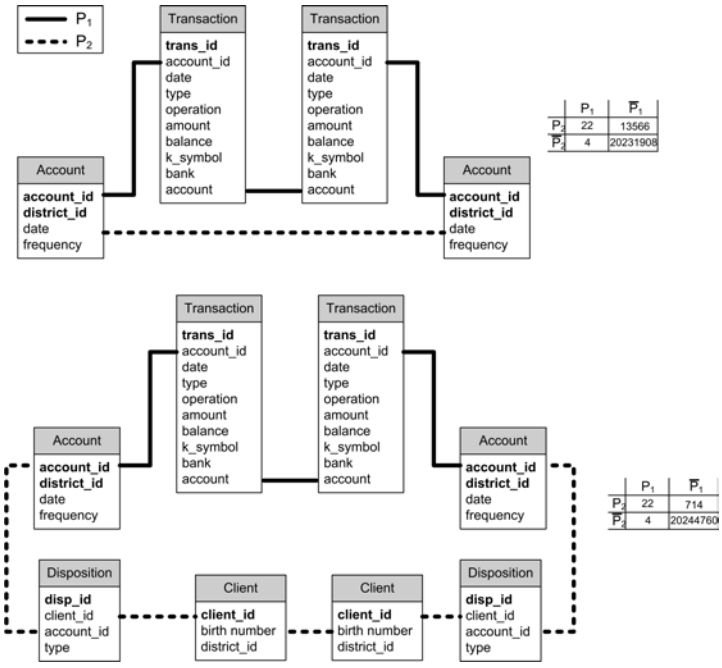

Fig. 12. Structure and contingency tables of pseudo–constraints PC_Bank$_6$:"*two (distinct) accounts linked by some transactions to the same third party account have been created on the same date*" (with $\rho = 0.7$ it has $d(\widehat{\boldsymbol{p}}) = 0.146$); PC_Bank$_7$:"*two (distinct) accounts linked by some transactions to the same third party account are held by clients with the same gender and birth date*" (with $\rho = 0.7$ it has $d(\widehat{\boldsymbol{p}}) = 0.146$).

Account_1			**Client_1**			**Account_2**			**Client_2**		
id	**dis.**	**date**	**id**	**birth–n.**	**dis.**	**id**	**dis.**	**date**	**id**	**birth–n.**	**dis.**
19	21	950407	25	395423	21	7418	21	950407	9196	395423	21
25	68	960728	31	620209	68	7424	68	960728	9202	620209	68
38	19	970808	46	405130	19	7437	19	970808	9217	405130	19
205	1	971008	246	570508	1	7606	21	950315	9417	490308	21
2062	70	960306	2501	406204	70	9422	70	960306	11596	406204	70
2064	54	961230	2503	775923	54	9424	54	961230	11598	775923	54
2073	59	970814	2514	570211	59	9433	59	970814	11609	570211	59
...	...	...	...	...	...	...	...	...	...	...	...

Fig. 13. Some couples of accounts (Account_1 and Account_2) linked by some transactions to the same third party account, along with data of their owners (Client_1 and Client_2 respectively)

if a candidate pseudo–constraint does not satisfy the significance test. Table 14 reports the statistics relative to an exhaustive search of cycle pseudo-constraints on the Czech bank database. In the reported experiment, we set the maximum cycle length to 4; the algorithm generated about 60000 candidate pseudoconstraints, out of which the tool found 60 pseudo–constraints with violations; the whole process took a little less than 2 hours.

Max cycle length l	4
Approximate number of candidates	60000
Number of pseudo–constraints with violations	60
Average search time per candidate	0.1 sec
Total computation time	1.7 h

Fig. 14. Statistics on the mining cycle pseudo–constraints in the Czech bank database

The data in Table 14 provide an idea of the typical pseudo–constraints mining experiment and they highlight interesting aspects. First, even if candidates pseudo constraints are many, only very few of them turn out to satisfy the test. Some of them have no violations at all and then the designer should be tempted to consider them as real constraints. The ones that have violations are normally interesting and sometimes hard to explain. Computational time is not prohibitive, because the search for pseudo–constraints should be done once for a given database; instead, instead, the search for violation instances of a given constraint for a given database state requires computing a given query only once.

8. RELATED WORK

Pseudo-constraints are a new and general pattern for data mining, so it is not surprising that it has relationships with a variety of data mining concepts and approaches, including relational data mining, deviation detection, association rules (and their variants such as causal and correlation rules), Bayesian Networks (for describing statistical models over typed relational domains), and finally approximated functional dependencies. Each such approach is next reviewed, outlining its essential tracts and the differences with pseudo-constraints.

8.1 Relational data mining

Pseudo–constraint is a data mining concept, falling in the *relational* (or *multi–relational*) *data mining* field [Džeroski 2003], the branch of KDD focusing on data mining in databases whose information is modeled with several types of objects. Specifically, *link discovery* [Mooney et al. 2002], which is part the emerging area of *link mining* [Getoor and Diehl 2005], faces the task of finding interestingly linked entity instances in multi–relational datasets, in order to identify abnormal or threatening activities. Link mining aims, among other things, at predicting the type of links between two objects or at inferring the existence of a link (which is called *link prediction*; see [Liben-Nowell and Kleinberg 2003] for a survey on link prediction in social networks), usually with a statistical approach.

Link mining is commonly accomplished via supervised learning of interesting paths; nearer to ours is the approach of *unsupervised link discovery* in [Shou–de Lin and Chalupsky 2003], where the interesting links between two instances, discovered by the method, are the infrequent ones.

In [Popescul and Ungar 2003] a novel statistical relational learning method is employed; in [Taskar et al. 2004] the framework of *relational Markov networks* is exploited; in [Getoor et al. 2003] probabilistic relational models are extended to include links. Some analysis methods include the search of the existence of a link after having observed another type of link, thus closely resembling our pseudo–

constraint concept.

Graph–based relational learning tackles the general issue of discovering common relational patterns (i.e., subgraphs) in large graphs such as a relational database instance; several tools support this approach, including SUBDUE [Cook and Holder 2000]. One could consider cycle pseudo–constraint as a particular pattern, stating that when two entity instances are linked in a way, they are also linked the other way round in an ER schema cycle.

These approaches are much broader than ours, thus being much less sharp and efficient on the particular link correlation we are interested in.

8.2 Deviation detection

Deviation detection or *outlier mining* (see for instance [Arning et al. 1996] or [Qi and Wang 2004]) is the branch of data mining primarily concerned with the identification of anomalous data, in order to cleanse data or to reveal improper behavior; on the practical side these researches are part of *crime data mining* [Chen et al. 2004], *fraud detection* [Bolton and Hand 2002], *intrusion detection* [Axelsson 2000], and so on. Many different techniques are employed, from statistics to supervised learning, the former often suffering from high false positive rate, the latter suffering from scarce availability of positive training examples and high need of costly human expert support.

Anyway, most of these techniques are driven by the pragmatics of the considered problem and do not take advantage of the relational structure of data, despite relationships between entities are widely recognized as a relevant parameter for suspiciousness.

8.3 Association rules

Pseudo–constraints have some resemblance with *association rules* [Agrawal et al. 1993], [Agrawal and Srikant 1994]: both state that when something holds usually something else holds too. However, association rules and pseudo–constraints are very different.

The first difference is that pseudo–constraints are assertions at the schema level (i.e., they are structurally similar to integrity constraints) while association rules are assertions about instances, typically searched within flat market baskets data (a market basket is a set of items bought together by a customer in, e.g., a department store). Thus, pseudoconstraints have an arbitrary structure and normally deal with multiple relations. Multi-table extensions of association rules via conjunctive queries have been proposed in [Dehaspe and Toivonen 2001] and [Goethals and den Bussche 2002], with some interesting preliminary results.

Even when restricted to market basket predicates, the two kinds of rules have different intended meanings. The importance of an association rule is measured by its support and confidence, which are statistical metrics about the association rule in the selected database instance. Given two predicates P_1 and P_2, the association rule "$P_1 \Rightarrow P_2$" states that (i) a good percentage of the instances that satisfy P_1 also satisfy P_2 (=high *confidence*) and (ii) a good percentage of the total number of instances satisfy both P_1 and P_2 (=high *support*). In general, there is nothing unusual in the instances that violate an association rule, i.e., there can be plenty of instances satisfying P_1 and not P_2. Pseudo–constraints, instead, are usually much

stronger rules and their model is such that their violation instances often need further explanation because they seem to break casual relationships. To illustrate this point with a concrete example, the classical association rule "who buys bread and butter also buys jam" is unlikely to be a pseudo–constraint because there are still lots of people who buy bread and butter but not jam, as buying bread and butter only slightly increases the probability of buying jam.

Another important difference is the statistical background of the two approaches. An association rule is a statement on a database instance; the idea is, of course, that what has been stated for a database instance should be inductively valid also in the real domain, but there is no formal statistical treatment of this induction. The pseudo–constraint approach, instead, is inherently statistical: a pseudo–constraint is an assertion on probabilities describing reality, and the method to test a candidate pseudo–constraint is a statistical test whose significance level has been studied (Section 5.2).

In short, association rules and pseudo–constraints have different scope, different meaning, different applications, and different parameters of evaluation.

8.4 Causal and correlation rules

[Silverstein et al. 2000] consider the problem of mining *causal relationships* in market baskets, instead of mere associations. For instance, the rule "*who buys hamburgers also buys barbecue sauce*" might be a *causal* rule (people buy barbecue sauce *because* they have bought hamburgers) while the rule "*who buys hot dogs also buys hamburgers*" might be only an *association* rule[2]. The appearance of causality in market basket mining research moves it somehow nearer to our approach, but the differences that were summarized in the previous section still hold; in addition, while causal rules try to give a deeper insight on the causes that motivate normal behaviors, pseudo-constraints are really concerned with abnormal - hence interesting - behaviors.

Association rules have been extended to *correlation rules* in [Silverstein et al. 1997][3]. A correlation rule states that the items of some set are *correlated* (better, "not independent"), meaning that the presence of an item in a transaction provides some information on the presence of the other item. The differences between association and correlation rules are, therefore, that the latter are non–directional, they can take into account negative correlation (i.e., "when some item occurs another is likely not to occur") and they state real dependencies among data (while an association rule could associate absolutely independent data).

Correlation rules are therefore nearer than association rules to our approach. But the differences are still many, the main ones being that correlation rules are not expressed at the schema level, they are not directional, they are not as "strong" as pseudo–constraints, and they are not violation–instances oriented. To illustrate the difference again with a concrete example, a correlation rule could show a "positive

[2]The problem of *probabilistic causality* is a long standing one. We refer the interested reader to [Pearl 2000] and [Hitchcock 2002] as good starting points.

[3]The name "correlation rules" has been changed to "dependence rules" in [Silverstein et al. 1998] because the notion of correlation is not the well known probabilistic notion of correlation, and the new name is more appropriate; however, the original name is still most used and we maintain it.

dependence between the absence of the item *coffee* and the occurrence of the item *tea* in market baskets", but a pseudo–constraint reformulation of such rule ("*who does not buy coffee buys tea*") would not be an obvious pseudo-constraint.

8.5 Bayesian Networks

Bayesian networks [Neapolitan 2004] are graphical models that encode probabilistic relationships among variables of interest. They are characterized by a structure, which represents the relationships among the problem variables, and the parameters that quantify such relationships. The structure can be given by a domain expert or otherwise it can be inferred from the data.

Given the structure, the network parameters are computed from the data using one of the several available algorithms [Neapolitan 2004]. The resulting network can then be used both (i) to generate new data from the underlying, unknown, distribution or (ii) to predict the values of certain target variables.

When comparing Bayesian network to pseudo–constraints we note that the two approaches are very different. Bayesian networks aim at predicting values of unknown variables based on the values of known variables, thus constructing domain knowledge; in contrast, pseudo–constraints assume that the most relevant domain knowledge has already been encoded during the database design step and work on the schema to extract relations that are probably unexpected to the database designer and to the database users.

8.6 Approximated Functional Dependencies

We recall that, given a set of attributes R, a relation r over R, and two subset of attributes $X, Y \subset R$, a functional dependency $X \rightarrow Y$ holds in r, if all the pairs of tuples that agree on X also agree on Y. Approximated functional dependencies [Kivinen and Mannila 1995; King and Legendre 2003; Ilyas et al. 2004] are functional dependencies that *almost hold*, i.e., that hold when a percentage ϵ of the relation r is not considered. Therefore, functional dependencies and approximated functional dependencies define implications over the domain that the relation r represents, in the form of implications that bind attribute values inside a relation. In approximated functional dependencies errors are equated to noise in the data and the error threshold ϵ [Kivinen and Mannila 1995; King and Legendre 2003; Ilyas et al. 2004] represents the amount acceptable noise.

Approximate functional dependencies are constraints of a particular format, hence they can be considered as a special case of pseudo–constraints, which express more general relations regarding the database. However, approximated functional dependencies cannot express typical pseudo–constraints when they represent the existence of arbitrary concepts, related through semantic relationships, arbitrary predicate expressions, and their compositions, as they are syntactically limited to equality predicates among attribute values.

Moreover, even when P_1 and P_2 could be expressed as functional dependencies, the definition of pseudo–constraints requires not only that P_2 depends upon P_1 (i.e., $\mathbb{P}[P_2|P_1] \simeq 1$, Equation 1), but also that, when P_1 is true, the probability that P_2 is also true increases (i.e., $\mathbb{P}[P_2|P_1] \gg \mathbb{P}[P_2]$, Equation 1). This second condition adds to pseudo–constraints a dimension which is not captured by approximate functional dependencies.

9. CONCLUSIONS

In this paper we introduced a new data mining pattern, named *pseudo–constraint*, that grasps the intuitive idea of a constraint which does not exactly holds, so that it has few and interesting violations. Integrity constraints which admit a limited number of violations have never been observed in this very general way in data mining, in spite of their intuitive simplicity, of their widespread presence, and of their clear interest. The first contribution of our work is therefore to point out a new and interesting concept. The rest of the contribution is the thorough exploration of some aspects of such concept.

—We have provided a *formal definition* of the notion of pseudo–constraint. The definition is an easily understood yet nontrivial *probabilistic model*, that captures the features underlying the intuitive idea of pseudo–constraint. The parameters used in the probabilistic model of pseudo–constraints are probabilities that refer to the real world mirrored by the database rather than the database instance itself. This gives to a pseudo–constraint the status of an assertion on reality, while most data mining patterns are assertions on database instances.

—We have determined an ad-hoc *statistical test* to validate the hypothesis that a candidate pseudo–constraint is a real pseudo–constraint. The test has a preset asymptotic significance level α. In spite of its nontrivial deduction, this test is extremely easy to perform.

—We have pointed out a special class of pseudo–constraints, the *cycle pseudo–constraints*, which are of major interest as they often occur in databases (as confirmed by our experimental results).

—We have designed a method to perform the *mining* of cycle pseudo–constraints and their violations. The method involves the generation of very simple SQL queries, whose execution can benefit from reuse of results of previously issued queries and from the main memory caching of query results.

—We have conducted extensive experiments in order to empirically validate our approach. The importance of a data mining pattern is in the number and importance of its occurrences in real databases: pseudo–constraints and their violations have proved widespread.

Our future work aims at consolidating the prototype tools already developed for detecting cycle pseudo–constraints and at identifying other important classes of pseudo–constraints that can be automatically detected.

REFERENCES

Agrawal, R., Imielinski, T., and Swami, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the* 1993 *ACM SIGMOD International Conference on Management of Data*. Washington, DC, USA.

Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the* 20*th International Conference on Very Large Data Bases*. Morgan Kaufmann, 487–499.

Arning, A., Agrawal, R., and Raghavan, P. 1996. A linear method for deviation detection in large databases. In *Proceedings of the* 2*nd International Conference on Knowledge Discovery and Data Mining*. 164–169.

Axelsson, S. 2000. Intrusion detection systems: A survey and taxonomy. Tech. Rep. 99–15, Chalmers University of Technology, Göteborg, Sweden.

Bolton, R. J. and Hand, D. J. 2002. Statistical fraud detection: A review. *Statistical Science 17,* 3, 235–255.

Ceri, S. and Widom, J. 1990. Deriving production rules for constraint maintenance. In *Proceedings of the* 16*th International Conference on Very Large Data Bases*. Morgan Kaufmann, 566–577.

Chen, H. et al. 2004. Crime data mining: A general framework and some examples. *IEEE Computer 37,* 4, 50–56.

Cook, D. J. and Holder, L. B. 2000. Graph–based data mining. *IEEE Intelligent Systems 15,* 2, 32–41.

Correa, J. C. 2001. Interval estimation of the parameters of the multinomial distribution. *Interstat*.

Dehaspe, L. and Toivonen, H. T. T. 2001. Discovery of relational association rules. In *Relational Data Mining*, S. Dzeroski and N. Lavrac, Eds. Springer.

Džeroski, S. 2003. Multi–relational data mining: An introduction. *SIGKDD Explorations Newsletter 5,* 1, 1–16.

Garcia-Molina, H., Ullman, J. D., and Widom, J. D. 2001. *Database Systems: The Complete Book*. Prentice Hall.

Getoor, L. and Diehl, C. P. 2005. Christopher p. diehl. *SIGKDD Explorations 7,* 2, 3–12.

Getoor, L., Friedman, N., Koller, D., and Taskar, B. 2003. Learning probabilistic models of link structure. *Journal of Machine Learning Research 3*, 679–707.

Goethals, B. and den Bussche, J. V. 2002. Relational association rules: getting Warmer. In *Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining*, D. J. Hand, N. M. Adams, and R. J. Bolton, Eds. Lecture Notes in Computer Science, vol. 2447. Springer, 125–139.

Hitchcock, C. R. Fall 2002. Probabilistic causation. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed.

Ilyas, I. F., Markl, V., Haas, P. J., Brown, P., and Aboulnaga, A. 2004. Cords: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD Conference*, G. Weikum, A. C. König, and S. Deßloch, Eds. ACM, 647–658.

Kendall, M. G. and Stuart, A. 1963. *The Advanced Theory of Statistics*, Second ed. Vol. 1. Charles Griffin & Company Limited, London.

King, R. S. and Legendre, J. J. 2003. Discovery of functional and approximate functional dependencies in relational databases. *Journal of Applied Mathematics and Decision Sciences 7,* 1, 49–59.

Kivinen, J. and Mannila, H. 1995. Approximate inference of functional dependencies from relations. *Theoretical Computer Science 149,* 1, 129–149.

Liben-Nowell, D. and Kleinberg, J. 2003. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*. New Orleans, LA, USA, 556–559.

Shou–de Lin and Chalupsky, H. 2003. Unsupervised link discovery in multi–relational data via rarity analysis. In *Proceedings of the Third IEEE International Conference on Data Mining*. Melbourne, FL, USA.

Mooney, R. J. et al. 2002. Relational data mining with inductive logic programming for link discovery. In *Proceedings of the National Science Foundation Workshop on Next Generation Data Mining*. Baltimore, MD, USA.

Neapolitan, R. E. 2004. *Learning Bayesian Networks*. Prentice Hall.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Popescul, A. and Ungar, L. H. 2003. Statistical relational learning for link prediction. In *Proceedings of the Workshop on Learning Statistical Models from Relational Data at IJCAI 2003*. Acapulco, Mexico.

Qi, H. and Wang, J. 2004. A model for mining outliers from complex data sets. In *Proceedings of the* 2004 *ACM Symposium on Applied Computing*. ACM Press, 595–599.

Shao, J. 1999. *Mathematical Statistics*. Springer.

	1	2	
1	$n_{1,1}$	$n_{1,2}$	$n_{1,\cdot}$
2	$n_{2,1}$	$n_{2,2}$	$n_{2,\cdot}$
	$n_{\cdot,1}$	$n_{\cdot,2}$	n

Fig. 15. Contingency table

Silverstein, C., Brin, S., and Motwani, R. 1998. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery 2,* 1, 39–68.

Silverstein, C., Brin, S., Motwani, R., and Ullman, J. D. 2000. Scalable techniques for mining causal structures. *Data Mining and Knowledge Discovery 4,* 2/3, 163–192.

Silverstein, C., Motwani, R., and Brin, S. 1997. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Tucson, AZ, USA.

Taskar, B., Wong, M.-F., Abbeel, P., and Koller, D. 2004. Link prediction in relational data. In *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. MIT Press.

Ullman, J. D. 1980. *Principles of Database Systems*. Computer Science Press.

Widom, J. and Ceri, S. 1995. *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann.

Żytkow, J. M. and Rauch, J., Eds. 1999. *Principles of Data Mining and Knowledge Discovery: Third European Conference, PKDD99*. Springer, Prague, Czech Republic.

A. STATISTICAL TEST FOR PSEUDO–CONSTRAINTS

In this appendix we provide details of the derivation of the test for pseudo–constraints. Knowledge of some advanced statistical topics is a prerequisite to understand the following discussion. The interested reader can find the background material, e.g., in [Shao 1999].

We remind that we have to find a test for the hypothesis:

$H_0 \quad d(\boldsymbol{p}) \leq 0$
$H_A \quad d(\boldsymbol{p}) > 0$

where $\boldsymbol{p} = (p_{11}, p_{12}, p_{21})$ is the parameter vector of the *multinomial distribution* of whom we have an observation summarized in a contingency table like the one in Figure 15 (or, which is the same for our purposes, the parameter vector of a *bivariate Bernoulli distribution* of whom we have an observation of a random sample of dimension n) and where:

$$d(\boldsymbol{p}) = \frac{p_{11}}{p_{11} + p_{21}} - \rho - (1 - \rho) \cdot (p_{11} + p_{12}) \tag{8}$$

Intuition. To decide whether $d \leq 0$ we basically see whether its maximum likelihood estimate $\widehat{d}_n = d(\widehat{\boldsymbol{p}}_n)$ is less than 0; but maximum likelihood estimates are not perfect: $\widehat{d}_n$ could be > 0 despite $d \leq 0$, leading us to erroneously reject the hypothesis; as we want to be cautious in rejecting the hypothesis, we actually accept it also if $\widehat{d}_n$ is a little larger than zero, but the more we trust our maximum likelihood estimate, the smaller is our "tolerance" for values of $\widehat{d}_n$ larger than zero. Specifically, we accepted the hypothesis for $\widehat{d}_n < z_\gamma \sigma_n(\widehat{\boldsymbol{p}}_n)$, where z_γ is a measure of how much we want to be "cautious" (and is related to the desired

significance level α^*) and $\sigma_n(\widehat{\boldsymbol{p}}_n)$ is a suitable inverse measure of the reliability of $\widehat{d}_n$.

Formal definition. The *proposed test* is:

$$\mathsf{Test}(\alpha^*, n_{11}, n_{12}, n_{21}, n) = \begin{cases} \textit{accept } H_0 & \text{if } h_{\gamma n}(\widehat{\boldsymbol{p}}_n) < 0 \\ \textit{reject } H_0 & \text{otherwise} \end{cases} \tag{9}$$

where

$$\widehat{\boldsymbol{p}}_n = (\widehat{p}_{n11}, \widehat{p}_{n12}, \widehat{p}_{n21}) = (\frac{n_{11}}{n}, \frac{n_{12}}{n}, \frac{n_{21}}{n})$$

and

$$h_{\gamma n}(\boldsymbol{p}) = d(\boldsymbol{p}) - z_\gamma \sigma_n(\boldsymbol{p})$$

and $\gamma = 1 - \alpha^*$ and z_γ is the γ–th quantile of the standard normal distribution,[4] and

$$\sigma_n(\boldsymbol{p}) = \frac{\sigma(\boldsymbol{p})}{\sqrt{n}}$$

and

$$\sigma(\boldsymbol{p}) = \sqrt{\frac{p_{11}p_{21}(2p_{11}(\rho-1) + 2p_{21}(\rho-1) + 1)}{(p_{11}+p_{21})^3} - (p_{11}^2 + p_{11}(2p_{12}-1) + p_{12}^2 - p_{12})(\rho-1)^2}$$

As demonstrated next, the significance level α of the test turns out to be approximately α^* if n is large enough.

A.1 Asymptotic distribution of $\widehat{D}$

Let the random vector $\widehat{\boldsymbol{P}}_n$ be the *maximum likelihood estimator* of the parameter vector $\boldsymbol{p} = (p_{11}, p_{12}, p_{21})$ based on the Bernoulli random sample of dimension n whose observation is summarized in the contingency table. We have:

$$\widehat{\boldsymbol{P}}_n = \left(\frac{N_{11}}{n}, \frac{N_{12}}{n}, \frac{N_{21}}{n}\right) \tag{10}$$

From the *asymptotic efficiency* of maximum likelihood estimators we have that, for all $\boldsymbol{p} \in \{(p_{11}, p_{12}, p_{21}) | p_{ij} \in (0,1),\ p_{11} + p_{12} + p_{21} < 1\}$:

$$\sqrt{n}(\widehat{\boldsymbol{P}}_n - \boldsymbol{p}) \xrightarrow[n\to\infty]{\mathcal{D}} \mathcal{N}(0, I(\boldsymbol{p})^{-1}) \tag{11}$$

where the *Fisher information* is:

$$I(\boldsymbol{p}) = E_{\boldsymbol{p}}[(\nabla \log L(\boldsymbol{p}; \boldsymbol{X}))(\nabla \log L(\boldsymbol{p}; \boldsymbol{X}))'] \tag{12}$$

where the *likelihood function* $L(\boldsymbol{p}; \boldsymbol{x})$ is:

$$L(p_{11}, p_{12}, p_{21}; x_{11}, x_{12}, x_{21}, x_{22}) = p_{11}^{x_{11}} p_{12}^{x_{12}} p_{21}^{x_{21}} (1 - p_{11} - p_{12} - p_{21})^{x_{22}} \tag{13}$$

Applying the *Delta method* to (11) and (8) we gain:

$$\sqrt{n}[d(\widehat{\boldsymbol{P}}_n) - d(\boldsymbol{p})] \xrightarrow[n\to\infty]{\mathcal{D}} \mathcal{N}(0, \nabla d(\boldsymbol{p})' \cdot I(\boldsymbol{p})^{-1} \cdot \nabla d(\boldsymbol{p})) \tag{14}$$

[4] For the typical value $\alpha = 0.05$ we have $z_\gamma \approx 1.645$.

for all $\boldsymbol{p} \in \{(p_{11}, p_{12}, p_{21}) | p_{ij} \in (0,1),\ p_{11} + p_{12} + p_{21} < 1\}$.

If we define:

$$\sigma(\boldsymbol{p}) = \sqrt{\nabla d(\boldsymbol{p})' \cdot I(\boldsymbol{p})^{-1} \cdot \nabla d(\boldsymbol{p})} \tag{15}$$

an easy calculation from (8), (12) and (13) shows that:

$$\sigma(\boldsymbol{p}) = \sqrt{\frac{p_{11}p_{21}(2p_{11}(\rho-1) + 2p_{21}(\rho-1) + 1)}{(p_{11}+p_{21})^3} - (p_{11}^2 + p_{11}(2p_{12}-1) + p_{12}^2 - p_{12})(\rho-1)^2} \tag{16}$$

and (14) is written:

$$\sqrt{n}[d(\widehat{\boldsymbol{P}}_n) - d(\boldsymbol{p})] \xrightarrow[n\to\infty]{\mathcal{D}} \mathcal{N}(0, \sigma^2(\boldsymbol{p})) \tag{17}$$

From the *consistency property* of maximum likelihood estimators we have that:

$$\widehat{\boldsymbol{P}}_n \xrightarrow[n\to\infty]{\mathcal{P}} \boldsymbol{p}$$

and so from the *continuous mapping theorem* for convergence in probability we obtain:

$$\sigma(\widehat{\boldsymbol{P}}_n) \xrightarrow[n\to\infty]{\mathcal{P}} \sigma(\boldsymbol{p}) \tag{18}$$

so that $\sigma(\widehat{\boldsymbol{P}}_n)$ is a consistent estimator of $\sigma(\boldsymbol{p})$.

By applying *Slutsky's theorem* to (17) and (18) we obtain, for all $\boldsymbol{p} \in \{(p_{11}, p_{12}, p_{21}) | p_{ij} \in (0,1),\ p_{11} + p_{12} + p_{21} < 1\}$:

$$\frac{d(\widehat{\boldsymbol{P}}_n) - d(\boldsymbol{p})}{\frac{\sigma(\widehat{\boldsymbol{P}}_n)}{\sqrt{n}}} \xrightarrow[n\to\infty]{\mathcal{D}} \mathcal{N}(0,1)$$

or

$$\frac{d(\widehat{\boldsymbol{P}}_n) - d(\boldsymbol{p})}{\sigma_n(\widehat{\boldsymbol{P}}_n)} \xrightarrow[n\to\infty]{\mathcal{D}} \mathcal{N}(0,1) \tag{19}$$

where we have defined:

$$\sigma_n(\cdot) = \frac{\sigma(\cdot)}{\sqrt{n}} \tag{20}$$

The *invariance property* of maximum likelihood estimators states that the maximum likelihood estimator $\widehat{D}_n$ of $d(\boldsymbol{p})$ based on a random sample of dimension n is:

$$\widehat{D}_n = d(\widehat{\boldsymbol{P}}_n) \tag{21}$$

and so from (19) we finally obtain:

$$\frac{\widehat{D}_n - d(\boldsymbol{p})}{\sigma_n(\widehat{\boldsymbol{P}}_n)} \xrightarrow[n\to\infty]{\mathcal{D}} \mathcal{N}(0,1) \tag{22}$$

for all $\boldsymbol{p} \in \{(p_{11}, p_{12}, p_{21}) | p_{ij} \in (0,1),\ p_{11} + p_{12} + p_{21} < 1\}$.

With some terminology abuse we could say that $\widehat{D}_n$ converges in distribution to a normal random variable with mean $d(\boldsymbol{p})$ and variance $\sigma_n^2(\widehat{\boldsymbol{P}}_n)$.

A.2 Approximate confidence interval for d

Our aim here is to find an *approximate one–tailed confidence interval* for d.[5] Formula (22) in the preceding subsection tells us that, for large enough n, the distribution of $\frac{\widehat{D}_n - d(\boldsymbol{p})}{\sigma_n(\widehat{\boldsymbol{P}}_n)}$ is approximately a standard normal distribution (we will later discuss in in A.4, by means of simulations, how large should n be for the approximation to be suitable for our purposes). We can therefore use $\frac{\widehat{D}_n - d(\boldsymbol{p})}{\sigma_n(\widehat{\boldsymbol{P}}_n)}$ as an asymptotic pivotal quantity to find approximate confidence intervals for $d(\boldsymbol{p})$.

Let z_γ be the γ–th quantile of the standard normal distribution (let's say $\gamma = 0.95$ so that $z_\gamma \approx 1.645$). For n large enough we have:

$$\mathbb{P}_{\boldsymbol{p}}\left[\frac{\widehat{D}_n - d(\boldsymbol{p})}{\sigma_n(\widehat{\boldsymbol{P}}_n)} < z_\gamma\right] \approx \gamma \tag{23}$$

for all $\boldsymbol{p} \in \{(p_{11}, p_{12}, p_{21}) | p_{ij} \in (0,1),\ p_{11} + p_{12} + p_{21} < 1\}$.

Let:

$$h_{\gamma n}(\cdot) \triangleq d(\cdot) - z_\gamma \sigma_n(\cdot) \tag{24}$$

From (21), (23) and (24) we obtain:

$$\mathbb{P}_{\boldsymbol{p}}\left[h_{\gamma n}(\widehat{\boldsymbol{P}}_n) < d(\boldsymbol{p})\right] \approx \gamma \tag{25}$$

for all $\boldsymbol{p} \in \{(p_{11}, p_{12}, p_{21}) | p_{ij} \in (0,1),\ p_{11} + p_{12} + p_{21} < 1\}$. So, when n is large enough $(h_{\gamma n}(\widehat{\boldsymbol{P}}_n), +\infty)$ is a one–tailed confidence interval for d whose confidence level is approximately γ.

A.3 Approximate test for $d \leq 0$

For our hypothesis:

H_0 $\quad d(\boldsymbol{p}) \leq 0$
H_A $\quad d(\boldsymbol{p}) > 0$

we adopt the following test:

—if $h_{\gamma n}(\widehat{\boldsymbol{p}}_n) < 0$ then *accept* H_0
—otherwise *reject* H_0

where $\gamma = 1 - \alpha^*$ and α^* is the desired level of significance for the test.

The real significance level α of the test is:

$$\alpha = \sup_{\boldsymbol{p}:\ d(\boldsymbol{p}) \leq 0} \mathbb{P}_{\boldsymbol{p}}[0 \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)] \tag{26}$$

Now, for $\boldsymbol{p}$ such that $d(\boldsymbol{p}) \leq 0$ the event $d(\boldsymbol{p}) \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)$ contains the event $0 \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)$, and so, for $\boldsymbol{p}$ such that $d(\boldsymbol{p}) \leq 0$ we have:

$$\mathbb{P}_{\boldsymbol{p}}[0 \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)] \leq \mathbb{P}_{\boldsymbol{p}}[d(\boldsymbol{p}) \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)]$$

[5] In the statistical community interval estimation of the parameters of the multinomial distribution has been widely studied; [Correa 2001] is a survey on this topic.

```
SIMULATE(α*, ρ, N, h, k)
for n ∈ N do
      i := 0
      repeat
          generate a random model, i.e., a random parameter vector p_i
          if the model is such that H_0 is true then
          i := i + 1
          count := 0
          for j = 1 to k do
              generate a random sample S_ij of dimension n
                          from a density of parameter p_i
              apply the test to S_ij
              if the test accepts H_0 then count := count + 1
          calculate ᾱ_{p_i} = count / k
      until i = h
      calculate ᾱ_n = sup_{i∈{1,...,h}} ᾱ_{p_i}
```

Fig. 16. Simulation's pseudo–code

Therefore we have:

$$\sup_{\boldsymbol{p}:\ d(\boldsymbol{p})\leq 0} \mathbb{P}_{\boldsymbol{p}}[0 \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)] \leq \sup_{\boldsymbol{p}:\ d(\boldsymbol{p})\leq 0} \mathbb{P}_{\boldsymbol{p}}[d(\boldsymbol{p}) \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)] \tag{27}$$

But from Equation (25) we know that for n large enough $\mathbb{P}_{\boldsymbol{p}}[d(\boldsymbol{p}) \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)]$ approximately does not depend on $\boldsymbol{p}$ and is equal to $1-\gamma$, which is equal to α^*; hence its superior limit will be equal to its value for $\boldsymbol{p} = \overline{\boldsymbol{p}}$ such that $d(\overline{\boldsymbol{p}}) = 0$:

$$\sup_{\boldsymbol{p}:\ d(\boldsymbol{p})\leq 0} \mathbb{P}_{\boldsymbol{p}}[d(\boldsymbol{p}) \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)] \approx \mathbb{P}_{\overline{\boldsymbol{p}}}[0 \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)] \approx \alpha^* \tag{28}$$

From (27) and (28) we have:

$$\sup_{\boldsymbol{p}:\ d(\boldsymbol{p})\leq 0} \mathbb{P}_{\boldsymbol{p}}[0 \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)] \approx \alpha^* \tag{29}$$

From (26) and (29) we finally obtain that for n large enough the significance level α of the proposed test is:

$$\alpha \approx \alpha^*$$

A.4 Simulation results on the test

Our test relies on the assumption that the dimension n of the random sample is large enough for the significance α of the test to be close enough to a desired level α^*. Here we report the results of a simulation we did in order to empirically evaluate $\overline{\alpha}_n$.

The simulation pseudo–code is in Figure 16 and the parameters we used are summarized in Figure 17. Note that $\overline{\alpha}_{\boldsymbol{p}_i}$ is an estimate of $\mathbb{P}_{\boldsymbol{p}_i}[0 \leq h_{\gamma n}(\widehat{\boldsymbol{P}}_n)]$ and that $\overline{\alpha}_n$ is an estimate of the significance level α. The results of the simulation are reported in Figure 18.

For small values of n (let's say $\langle 1000$) the values of $\overline{\alpha}_n$ are actually much larger than α^*. This is due to the presence of parameter vectors $\boldsymbol{p}$ such that the following two conditions hold:

Parameter:	Meaning:	Value:
ρ	a parameter in the definition of pseudo–constraints	0.9
α^*	desired significance level	0.05
N	set of values of n to be investigated	$\{100, 400, 1600, 6400, 25600\}$
h	number of models with $d(\boldsymbol{p}) \leq 0$ generated	1000
k	number of samples generated for each model	100

Fig. 17. Parameters used in the simulation

n	Estimate of significance ($\overline{\alpha}_n$)
100	0.59
400	0.19
1600	0.02
6400	0.06
25600	0

Fig. 18. Simulation results

$p_{1,1}$	=	0.54114
$p_{1,2}$	=	0.44239
$p_{2,1}$	=	0.00299
d	=	−0.00384

Fig. 19. A borderline parameter vector

(1) $\boldsymbol{p}$ is near the frontier of the parameter space
(2) $d(\boldsymbol{p}) \approx 0$

These conditions are satisfied in Figure 19, and a simulation is reported in Figure 20. Indeed, if the parameter vector is near to the frontier of the parameter space, then the normal approximation needs large values of n in order to be reliable. For such parameters and for small n, the variance of $d(\widehat{\boldsymbol{p}}_n)$ is not suitably counterbalanced by $z_\gamma \sigma_n(\widehat{\boldsymbol{p}}_n)$ and so, if $d \leq 0$ but $d \approx 0$ then $d(\widehat{\boldsymbol{p}}_n) - z_\gamma \sigma_n(\widehat{\boldsymbol{p}}_n)$ will often be ≥ 0, leading to an erroneous reject.

But if we bound the distance of $\boldsymbol{p}$ from the frontier to be larger than some value, then the normal approximation is good also for comparatively small values of n and the test is reliable for every d (see Figure 21 and Figure 22). On the other hand, if $\boldsymbol{p}$ is near the frontier but $d(\boldsymbol{p})$ is much less than 0, then the approximation is not reliable, but nevertheless the test does not fail because also $d(\widehat{\boldsymbol{p}}_n)$ will usually be much less than 0. Anyway, the value of n in real applications is usually very high, at least for containment pseudo–constraints (as it is the product of the cardinalities of two entities, which in real databases are usually high themselves), and therefore the test is reliable also for almost-borderline parameter values.

n	Estimate of the probability of rejection
400	0.2792
1600	0.0076
6400	0
25600	0

Fig. 20. Simulation results for the parameter of Figure 19

n	Estimate of the probability of rejection
100	0.25
400	0.05
1600	0
6400	0.02
25600	0

Fig. 21. Simulation results for $\boldsymbol{p}$'s distance from the frontier > 0.005

n	Estimate of the probability of rejection
100	0.02
400	0
1600	0
6400	0
25600	0

Fig. 22. Simulation results for $\boldsymbol{p}$'s distance from the frontier > 0.05