

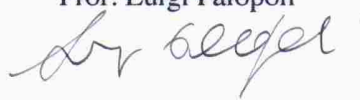
UNIVERSITÀ DELLA CALABRIA
Facoltà di Ingegneria
Dipartimento di Elettronica, Informatica e Sistemistica

Dottorato di Ricerca in Ingegneria dei Sistemi ed Informatica
XXIV ciclo
Settore Scientifico Disciplinare ING/INF-05

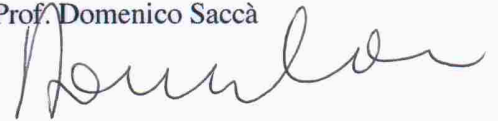
Tesi di Dottorato

**The Generative Aspects of Count Constraints:
Complexity, Languages and Algorithms**

Coordinatore
Prof. Luigi Palopoli



Supervisore
Prof. Domenico Saccà



Edoardo Serra


Anno Accademico 2010/2011

UNIVERSITÀ DELLA CALABRIA
Facoltà di Ingegneria
Dipartimento di Elettronica, Informatica e Sistemistica

Dottorato di Ricerca in Ingegneria dei Sistemi ed Informatica
XXIV ciclo
Settore Scientifico Disciplinare ING/INF-05

Tesi di Dottorato

The Generative Aspects of Count Constraints: Complexity, Languages and Algorithms

Coordinatore
Prof. Luigi Palopoli

Supervisore
Prof. Domenico Saccà

Edoardo Serra

Anno Accademico 2010/2011

to my Mother

Acknowledgments

I owe my deepest gratitude to my mentor Prof. Domenico Saccà who, under his direction, has been able to bring out the best of me and give me the right space to take confidence with my capabilities, without ever suppress my creativity.

I wish to express my gratitude to Dr. Antonella Guzzo and Dr. Luigi Moccia (Antonella and Luigi) that with their suggestions always help me, proving to be true friends.

Special thanks to Prof. Carlo Zaniolo that during my visit to UCLA (Los Angeles, CA) has been present and under his supervision I spent a productive year.

I wish to thank Prof. Francesco Scarcello (Francesco) whose door are always opened for any theoretical question.

Last but not the least I would like to thank my Dad and my girlfriend Francesca.

Rende,

November 2011

Edoardo Serra

Contents

1	Introduction	1
1.1	Inverse Frequent set Mining (IFM)	2
1.2	Count constraints and the inverse OLAP problem	6
1.3	Datalog with frequency support goals	9
1.4	Contributions of the thesis	11
1.5	Organization of the thesis	12
2	Inverse Frequent Itemset Mining Problem (IFM)	15
2.1	Introduction	15
2.2	Problem Formalization and Complexity Issues	16
2.3	A Level-Wise Solution Approach for IFM_S	19
2.3.1	Description of the Algorithm	19
2.3.2	Optimization Issues	21
2.3.3	An Example of computation	22
2.4	Solving the General Case of IFM	23
2.5	Computational Result	24
2.5.1	Test instance	25
2.5.2	Results	25
3	A New Generalization of IFM	29
3.1	Introduction	29
3.2	Problem Formalization	32
3.2.1	The $\kappa\text{-IFM}_{\sigma'}$ as an Integer Linear Program	35
3.2.2	The $\kappa\text{-IFM}_{\sigma'}$ as a Linear Program	38
3.3	Column Generation Algorithm to solve LP	39
3.3.1	Column Generation Algorithm	39
3.3.2	Complexity of the Pricing Problem	41
3.3.3	A Heuristic Algorithm for the Pricing Problem	44
3.3.4	Heuristic Column Generation Algorithm	45
3.4	Computational results	47
3.4.1	Test instances	47

3.4.2	Results	50
3.5	Related Works	55
3.5.1	Solving FREQSAT with Heuristic Column Generation Algorithm	57
3.5.2	Comparison with IPF	58
3.6	Applications of IFM	59
4	Count Constraints and the Inverse OLAP Problem	61
4.1	Introduction	61
4.2	Preliminaries and related work	63
4.2.1	From IFM to Inverse OLAP	63
4.2.2	Data Exchange	64
4.3	Count Constraints	64
4.3.1	A Motivating Example	68
4.4	The Inverse OLAP Problem: Definition and Complexity	70
4.4.1	Binary Domain Inverse OLAP	70
4.4.2	Binary Attribute Inverse OLAP	77
4.4.3	Data Complexity of Inverse OLAP	78
4.5	A Step towards Aggregate Data Exchange	79
5	Datalog with frequency support goals	83
5.1	Introduction	83
5.2	Related Work	84
5.3	Datalog ^{FS} by Examples	85
5.4	Semantics of Datalog ^{FS}	88
5.4.1	Rewriting of Datalog ^{FS} into Datalog	88
5.4.2	Stratified Datalog ^{FS}	89
5.4.3	Recursive Datalog ^{FS}	89
5.5	Multi-Occuring Predicates	90
5.6	Implementation & Optimization	93
5.6.1	Differential Fixpoint	93
5.6.2	Magic Sets	95
5.6.3	Avoiding Expansions	96
5.7	Scaling	98
5.8	More Advanced Applications	99
5.8.1	Diffusion Models with Datalog ^{FS}	100
5.8.2	Markov Chains with Datalog ^{FS}	102
6	Conclusions	107
	References	109

Introduction

Important elements of database models are the *integrity constraints*. The integrity constraints are used to enhance the expressiveness of data model; In other words the integrity constraints represent properties that the data structure of database must have. If the data satisfy the integrity constraints, such data are said *consistent*. A typical database model is the relational model, where the data structure is represented as a set of relations and each relation identifies an entity described through a set of attributes. In the relational schema there are more types of constraints as *functional dependencies*, *primary key*, *foreign key* and *attributes domain* [AHV95]. Consider a classical example of a database of a university where students attend courses and each course is done in a specific quarter of the year. A possible relational schema is the following:

$$student(\underline{idn}, name), attend(student, \underline{course}), course(\underline{id}, quarter)$$

and we have the following integrity constraints:

- **Functional dependencies:** Given a relation R , a set of attributes X in R is said to functionally determine another attribute Y in R , $X \rightarrow Y$, if, and only if, each X value is associated with precisely one Y value. A special case of functional dependency is the primary key constraint that imposes that each record of a relation is uniquely identified by a specific set of attributes, called key, i.e. in a relation cannot exist two records with the same assignment of values of the key set. In our example $course(\underline{id}, quarter)$ the key is the underline attribute id that can be express as functional dependency $id \rightarrow quarter$.
- **Foreign key constraints:** A foreign key imposes that a specific attribute in a relation has values take out of values of a key attribute in an other table, for example with the notations $attend(-, X) \rightarrow \exists Y : course(X, Y)$ we indicate that the values of attribute $course$ in relation $attend$ are at most the values in attribute id in relation $course$.
- **Attribute domain constraints:** An attribute domain constraint imposes that all value of a specific attribute in a relation must belong to a specific domain. For

example with the notation $course(-, Y) \rightarrow Y \in \{fall, winter, spring\}$ we imposes that each values of the attribute *quarter* in the relation *course* must belong to the set $\{fall, winter, spring\}$.

- **Cardinality constraint:** A cardinality constraint is a special constraints that imposes that the cardinality of a specific projection over a relation, belongs to a particular range of values. We recall that the projection operation extracts only the specified attributes from a tuple or set of tuples [AHV95]. In the example we have that every course must have at least 20 students and at most 25.

Common problems dealing with integrity constraints are: *data consistency checking* and *constraints consistency checking*. Since the integrity constraints imply that the data in the database must have a particular structure, the first problem consists in verifying if the data in the database satisfy the given constraints (this phase is also called *check phase*). Typically data consistency check is done after a change (update or insert) of the data, and the validation depends on the result of the check phase. When there is a lot of constraints (especially in the context of ontology), an other interesting problem consist in verifying if there exists at least a data structure that satisfy them, in such case the constraints are said consistent (constraints consistency checking). Actually, current literature focused on the decisional version of both two problems, while the generation of data has not been enough investigated. Conversely, we consider the constraints discussed above, from a different point of view, that is by investigating on the generative aspect, i.e. the way to materialize the data that satisfy them. Moreover, we focus on a special type of constraint over aggregate value, called *count constraint*, which is described by two elements: a description of the set in which the count aggregate is applied and a numeric interval in which the value of the count aggregate must belong. We will briefly show in the Section 1.2 how our count constraints are able to express all previous integrity constraints.

We consider the generative aspect of count constraint in the following three contexts:

1. Transaction database and support constraints (Section 1.1);
2. Relational database and count constraints (Section 1.2);
3. Deductive database: datalog with frequency support goals (Section 1.3).

1.1 Inverse Frequent set Mining (IFM)

Transaction databases are databases where each tuple, called *itemset* (or *transaction*), is defined as a subset of a fixed set of *items* \mathcal{I} (e.g. in market data, a transaction corresponds to one purchase event and each item is a salable product). Given an itemset J and a transaction database, two measure are defined: (i) *number of duplicates*, i.e. the number of times that the itemset appears in the transaction database, and (ii) *support*, i.e. the number of itemsets in the transaction database that contain the specified itemset. An example of transaction database over the items $\mathcal{I} = \{a, b, c, d\}$ is the following:

transaction	NDUP
$\{a, b, c\}$	8
$\{a, b, d\}$	14
$\{a, b\}$	4
$\{d\}$	2

where the field *transaction* represents the itemsets and NDUP are its numbers of duplicates for the itemsets. Over such database, given an itemset subset of $\{a, b, c, d\}$ we can compute its support; for example, the support of $\{a, b\}$ is equal to $8+14+4 = 26$ because $\{a, b\}$ is contained in $\{a, b, c\}$, $\{a, b, d\}$ and $\{a, b\}$.

The Inverse Frequent set Mining problem (IFM) [Mie03] is the following. Given a set S of frequent itemsets subset of \mathcal{I} and a specified range of support for each itemset in S , construct a transaction database D (if any exist) that satisfies a set of support constraints over S . A support constraint is a triple composed by an itemset I and two non negative integers σ_{min} and σ_{max} that represents the extremes of range $[\sigma_{min}, \sigma_{max}]$. Such support constraint is satisfied in a database D if the support of the itemset I belongs to $[\sigma_{min}, \sigma_{max}]$. It is important to notice that, since the support of an itemset is the number of transactions that contain this itemset, then a support constraints can be seen as a specialization of count constraint. Its major applications concern privacy preserving and generator for benchmarking data.

For example, consider the following instance of IFM over a set of items $\{a, b, c, d\}$:

I	σ_{min}	σ_{max}
$\{a, b, c\}$	8	10
$\{a, d\}$	10	14
$\{b, d\}$	18	20
$\{d\}$	20	40

where the field I represents the itemset and $[\sigma_{min}, \sigma_{max}]$ is the ranges of its support. The following two databases satisfy this constraints:

transaction	NDUP	transaction	NDUP
$\{a, b, c\}$	8	$\{a, b, c, d\}$	6
$\{a, b, d\}$	14	$\{a, b, c\}$	2
$\{b, d\}$	4	$\{a, b, d\}$	8
$\{d\}$	2	$\{b, d\}$	6

Normally, the itemsets provided in S are only the frequent ones, i.e. the itemsets whose support is greater or equal than a fixed threshold. However, with classical IFM formulation this assumption is not holds. In fact IFM formulation does not impose that each itemset not in S must be infrequent i.e. with a support below than the frequency threshold. As an example, we can see in the previous two databases, that the support of $\{a, b\}$ is 22 in the first database and 16 in the second. If we compare the support of $\{a, b\}$ with respect to the support of other itemset in S we cannot say that $\{a, b\}$ is infrequent. Therefore the itemset $\{a, b\}$ results frequent even if it is not specified. We consider this aspect as an anomaly of the formulation.

To solve this anomaly we initially define a new version of IFM, called IFM_S . Such formulation, under the assumption that each itemset not in S must be infrequent (not significant), imposes that each itemset not in S has support equal to zero or equivalently the number of duplicates is zero. We prove that this subproblem is NP-complete, and provide an heuristic algorithm that always satisfies the maximum support constraints, but that treats minimum support constraints as soft ones that are enforced as much as possible. In fact, a thorough experimentation evidences that minimum support constraints are hardly violated in practice, and that such negligible degradation in accuracy is compensated by very good scaling performances.

Despite the IFM_S solves this anomaly, the assumption that each itemset not in S must have a support equal to zero, can be restrictive. Consider the following instance over items $\mathcal{I} = \{a, b, c\}$.

I	σ_{min}	σ_{max}
$\{a, c\}$	4	4
$\{b, c\}$	3	3
$\{c\}$	5	5

Since the number of duplicates or the support of itemsets not in S must be equal to zero, then the itemset in D can only be the itemset in S . Thus, there is not solution to the IFM_S problem in fact the database that satisfies the previous constraints must have the following structure:

transaction	NDUP
$\{a, c\}$?
$\{b, c\}$?
$\{c\}$?

It is easy to see that, if we satisfy the first two constraints over itemsets $\{a, c\}$ and $\{b, c\}$, automatically the constraint over itemset $\{c\}$ is violated because its support is equal to $4 + 3 = 7$.

If we admit that can exist in the database other itemsets not defined in S but with limited support (in this case less or equal 2), then it is possible generate a database that satisfies the above constraints. Such database is the following:

transaction	NDUP
$\{a, b, c\}$	2
$\{a, c\}$	2
$\{b, c\}$	1

It is important to notice that the support of $\{a, b, c\}$ is less than of the one of other itemset in S , therefore it can be considered infrequent.

To tackle this problem we propose a *pregeneration* method. The *pregeneration* method consists to generate a new instance of IFM_S over an extended set of itemsets $S \cup S^+$ where S^+ is a the set of new itemsets different from S . The *pregeneration* method heuristically builds by using itemsets in S , a limited number of itemsets not in S that puts in S^+ . In the new instance of of IFM_S is imposed that the support of each itemset in S^+ is lesser or equal to a fixed support threshold σ' .

At this point it is important to show that not all itemsets not in S can be constrained. Consider, in fact, the following instance over a set of items $\{a, b, c\}$:

I	σ_{min}	σ_{max}
$\{a, c\}$	4	4
$\{c\}$	5	5

If we impose that the threshold of new itemsets σ' is equal to 2, then the itemset $\{a\}$, not contained in S , must have a support less or equal to 2, and therefore the itemset $\{a, c\}$ cannot have a support equal to 4 but only 2. For this reason, we constraint only the itemsets not in S such that does not exist an itemset in S that contains them, such set is called S'

$$S' = \{I \in U_{\mathcal{I}} \setminus S \mid \nexists I' \in S : I \subset I'\}$$

where $U_{\mathcal{I}}$ is the set of all itemsets contained in \mathcal{I} . We think that the information about of itemsets in S are more accurate w.r.t the succinct information given by the threshold support for the itemsets not in S . In Chapter 2 are reported the IFM_S formalization, the IFM_S heuristic, the pregeneration method, and their relative experimental results.

Nevertheless the experimental results of previous methods are satisfactory, the *pregeneration* of a limited set of itemset is not always sufficient, because it is difficult to establish what are the itemsets that can improve the satisfying of the constraints in S .

An alternative approach that significantly improve the previous results consists in a generation of itemsets not in S of the type step by step, i.e. starting with a fixed set of itemsets, solve the problem with such set and after try to find a new itemset that can improve the current solution. Such process continues until the current solution satisfies each constraint or does not exist an other itemset that improves the solution.

The big problem in such approach is that we must always guaranty that the support of itemsets in S' is less or equal than a fixed threshold. For this reason, we provide a new formulation called $\text{IFM}_{\sigma'}$, that imposes that each itemset not in S has a support less or equal than a fixed threshold σ' . We will show that this problem is NEXP-complete and such complexity is caused by imposing that each itemset not in S has a support less or equal to σ' . In fact to verify that a database satisfies all the constraints, we must compute for each itemset, its support and the number of each itemset in S' can be exponential in the size of \mathcal{I} .

Even if the monotonicity property of the support allows to verify only the support of the minimal itemsets in S' , i.e. the set $B_{S'}$ defined as follow:

$$B_{S'} = \{I \in S' \mid \nexists J \in S' : J \subset I\}$$

the cardinality of $B_{S'}$ can be exponential in the size of S . In fact, consider the following set of itemsets S over the items $\mathcal{I} = \{a_0, a_1, b_0, b_1, c_0, c_1\}$:

S
$\{b_0, b_1, c_0, c_1\}$
$\{a_0, a_1, c_0, c_1\}$
$\{a_0, a_1, b_0, b_1\}$

the set $B_{S'}$ is :

$B_{S'}$	
$\{a_0, b_0, c_0\}$	$\{a_0, b_0, c_1\}$
$\{a_0, b_1, c_0\}$	$\{a_0, b_1, c_1\}$
$\{a_1, b_0, c_0\}$	$\{a_1, b_0, c_1\}$
$\{a_1, b_1, c_0\}$	$\{a_1, b_1, c_1\}$

As it is possible to see, the cardinality of $B_{S'}$ is equal to 2^3 , where 3 is the number of the itemsets in S . In same way, it is possible to produce exponential examples ($|B_{S'}| = 2^n$) with n itemsets in S and $2 * n$ items.

Because $B_{S'}$ can still be exponential in the size of S , we individuate a parameter k relative to the number of items. If this parameter is bounded, then also the cardinality of $B_{S'}$ is bounded. Such parameter allows to define a parametrization of $\text{IFM}_{\sigma'}$, called $\kappa\text{-IFM}_{\sigma'}$, that can be reduced to the original IFM problem, thus they have the same complexity: NP-hard and PSPACE.

Moreover we will show that the $\text{IFM}_{\sigma'}$ can be formulated as an integer linear problem with huge number of variables and constraints, in general both exponential in the size of the input. At the same time it will showed that the parametric versions can be solved as an integer linear problem only with a huge number of variables.

In order to concretely solve $\kappa\text{-IFM}_{\sigma'}$, we relax the integer constraints over the number of duplicates and use the *column generation method* [DT03, DDS05] that is an efficient way to solve particular linear problem with a huge number of variables.

In order to use the column generation method, it is necessary solve its specific pricing problem. In our context the pricing problem is NP-complete and we solve this in exact way by an efficient integer linear formulation and a heuristic way by using a greedy dept-first algorithm.

The experimental results prove that our new approach is efficient and effective and the relaxed integer approximation does not compromise the quality of results. Moreover, despite the $\text{IFM}_{\sigma'}$ presents a strong complexity, the real life instances considered, present a very small value of the k parameter, and, therefore, they are tractable with our approach. The problem $\text{IFM}_{\sigma'}$ with its parametrization $\kappa\text{-IFM}_{\sigma'}$ and column generation methods are discuss Chapter 3.

1.2 Count constraints and the inverse OLAP problem

In the context of relational databases, we propose integrity constraints based on the first order predicate calculus. Such a constraint is represented by a rule of following form:

$$\forall \mathbf{X} (\alpha \rightarrow \beta_{min} \leq \#(\{ \mathbf{Y} : \gamma \}) \leq \beta_{max}).$$

The head of such rule represents an integer interval where the value of count aggregate $\#(\{ \mathbf{Y} : \gamma \})$ must belong, and the body α is a conjunction of atoms. For brevity, in this section we explain part of the syntax and semantics of our count constraints by some examples.

Consider the initial examples of integrity constraints over the schema:

$$student(idn, name), attend(student, course), course(id, quarter)$$

We can rewrite the previous constraints by using our count constraints:

- the functional dependency (the primary key) can be express as

$$\forall ID (D_{id}(ID) \rightarrow 0 \leq \#(\{Q : course(ID, Q)\}) \leq 1)$$

where D_{id} is the domain of attribute id in the relation $course$. Such constraint can be intuitively read as: for the each value of domain D_{id} there is at most one tuple with such a value of id .

- the foreign key can be expressed as

$$L = \#(\{ID : attend(-, ID)\}) \rightarrow L \leq \#(\{ID : attend(-, ID), course(ID, -)\}) \leq L$$

Such constraint can be intuitively read as: let L the number of id values of the relation $attend$ then the number of id of relation $attend$ that are also present in the id values of the relation $course$ is equal to L .

- in the same way of foreign key we can model the domain attribute constraint:

$$L = \#(\{Y : course(-, Y)\}) \rightarrow L \leq \#(\{Y : course(-, Y), Y \in \{fall, winter, spring\}\}) \leq L$$

- The cardinality constraint can be express as

$$\forall C (course(C, -) \rightarrow 20 \leq \#(\{S : attend(S, C)\}) \leq 25)$$

Such constraint can be intuitively read as: for each course C the number of students that attend the course C is between 20 and 25.

Actually, our language is more complex and provides the using of sets definitions. Consider the relational schema formed by only one relation $trans(tid, item)$ with domains D_{tid} and D_{item} . Such relation schema plus count constraints can be used to express the problem of Inverse Frequent Itemset over transaction Databases defined in Section 1.1. In fact the following transaction database can be transformed in an instance of the relation $trans$:

transaction	NDUP
{a, c}	2
{b, c}	1

 \Rightarrow

id	tid
1	a
1	c
2	a
2	c
3	b
3	c

where $D_{tid} = \{1, 2, 3\}$ and $D_{item} = \{a, b, c\}$. Therefore, if we want to describe some constraints over the support of itemsets we can use our count constraints with the set definition. For example, if we want to say that the support of itemset $\{a, b, c\}$ is between 4 and 6 we can use the following count constraint:

$$I = \{a, b, c\} \rightarrow 4 \leq \#(\{TID : I \subseteq \{ITEM : trans(TID, ITEM)\}\}) \leq 6$$

This constraint can be read in the following way: let I be the set $\{a, b, c\}$ then the number of tid (transactions) s.t. I is a subset of the set of items belonging to tid is between 4 and 6.

In the following, similarly to the problem of Inverse Frequent set Mining, we define the *inverse OLAP* problem where, instead of find a transaction database that satisfies a specific set of constraints over the support of the itemsets, we want to find a relational database that satisfies a set of our count constraints.

We focus on a relation schema typical in the world of OLAP [CD97, LL03, Han05], i.e. the *star schema* (see Figure 1.1) that consists of one fact table referencing any number of dimension tables.

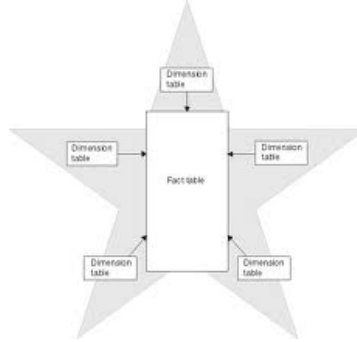


Fig. 1.1. Star schema

The fact table represents the relation where the values to aggregate are contained. In our case, we consider only count aggregates, and then the fact table is only a relation $R(A_1, \dots, A_n)$ with $n > 0$, where the domains D_1, \dots, D_n are specified. The dimension tables are configured as hierarchy domains, i.e. binary relation linked to some attribute in R . Thus, the inverse OLAP problem is the following: given a set of domains D_1, \dots, D_n with possibly a fixed additional number of hierarchy domains and a set of count constraints, decide whether there exists (or to find) a relational table $R(A_1, \dots, A_n)$ that satisfies the count constraints.

Similarly to the query problems in databases it is possible to consider for this problem several type of complexity [Var82]: data complexity (in input only the domains), program complexity (in input only count constraints) and combined complexity (in input domains and count constraints). We prove that inverse OLAP is NEXP-complete under data, program and combined complexity. Note that, since

our count constraints are very expressive (see the previous integrity constraints expressed with our count constraints), consider only star schema is not a limitation.

In Chapter 4 we describe in an extensive way the syntax and semantics of count constraints, the inverse OLAP problem, and its special cases with complexity analysis; in addition, in the last part of the chapter, some examples showing that our framework is step toward the data exchange problem with count constraints are described. We recall that this is the problem of migrating a data instance from a source schema to a target schema such that the materialized data on the target schema satisfies the integrity constraints specified by it.

We believe that the strict connection between inverse OLAP and IFM problem, can help us to export the efficient approach used for IFM in the case of inverse OLAP.

1.3 Datalog with frequency support goals

Similarly to the language of count constraints above presented, we introduce a simple extension of Datalog [Hel10, dMMAG, HGL11], called Datalog^{FS}, that enables us to query and reason about the number of distinct occurrences satisfying given goals, or conjunction of goals, called Frequency Support goal (or FS-goal), in rules. The form of FS-goal is the following:

$$K_j : [\text{expr}_j(X_j, Y_j)]$$

where K_j is the counting variable and $\text{expr}_j(X_j, Y_j)$ is a conjunction of positive atoms. Such count atom is satisfied, intuitively, if there exist K_j different satisfying ground instances of $\text{expr}_j(X_j, Y_j)$. In the following, we show a simple program in Datalog^{FS} that computes the persons that will come to a particular event.

$$\begin{aligned} \text{willcome}(X) &\leftarrow \text{sure}(X). \\ \text{willcome}(Y) &\leftarrow 3 : [\text{friend}(Y, X), \text{willcome}(X)]. \end{aligned}$$

The last rule is based on the assumption that a person will come to an event if at least three friends will come to this event. In fact, as it is possible to see by the FS-goal $3 : [\text{friend}(Y, X), \text{willcome}(X)]$, in the program we derive the predicate $\text{willcome}(y)$ if y has at least three different friends x_1, x_2, x_3 ($\text{friend}(y, x_1)$, $\text{friend}(y, x_2)$ and $\text{friend}(y, x_3)$) such that x_1, x_2, x_3 will come to the event ($\text{willcome}(x_1)$, $\text{willcome}(x_2)$ and $\text{willcome}(x_3)$).

In the previous program the counting variable K_j is fixed to the value 3. However, in a general program K_j is not fixed and can be present in other predicates in the body or directly present in head of the a rule, as in the last rule of the following example that computes the number of friends of a person.

$$\begin{aligned} &\text{friend}(x_0, x_1) \\ &\text{friend}(x_0, x_2) \\ &\text{friend}(x_0, x_3) \\ \text{nFriend}(X, K) &\leftarrow K : [\text{friend}(X, Y)]. \end{aligned}$$

In this case, the facts $\text{nFriend}(x_1, 1)$, $\text{nFriend}(x_1, 2)$ and $\text{nFriend}(x_1, 3)$ are derived. Actually, we are only interested in the fact $\text{nFriend}(x_1, 3)$ that contains the maximum value of the counting variable K . For this reason, we introduce the *m-predicates* (multiplicity predicates) that, instead of storing a fact for each value assumed by K , it directly stores the multiplicity fact with the maximum value of K . In this case the last rule of the previous program is rewritten by using an *m-predicates* as follows:

$$\text{nFriend}(X) : K \leftarrow K : [\text{friend}(X, Y)].$$

and the derived facts are only $\text{nFriend}(x_1) : 3$. It is important to notice that we do not change the semantics but only the way of computing. More specifically, the fact $\text{nFriend}(x_1) : 3$ is a compact way to represent the three facts $\text{nFriend}(x_1, 1)$, $\text{nFriend}(x_1, 2)$ and $\text{nFriend}(x_1, 3)$ i.e. if we know $\text{nFriend}(x_1) : 3$ then we also know $\text{nFriend}(x_1) : 2$ and $\text{nFriend}(x_1) : 1$.

Consider now the following program, where, given a complex object formed by a fixed number of subpart (a subpart can be a basic object or an other complex object), it able to compute the number of elementary part of the complex object. Let $\text{basic}(\text{part})$ be facts that describe the basic part and $\text{assbl}(\text{part}, \text{sub}, \text{qty})$ be the facts that describe the number (*qty*) of sub-part (*sub*) that are present in a complex object (*part*), the program is the following:

$$\text{cassb}(\text{Part}, \text{Sub}) : \text{Qty} \leftarrow \text{assbl}(\text{Part}, \text{Sub}, \text{Qty}).$$

$$\begin{aligned} \text{cbasic}(\text{Pno}) : 1 &\leftarrow \text{basic}(\text{Pno}). \\ \text{cbasic}(\text{Part}) : K &\leftarrow K : [\text{cassb}(\text{Part}, \text{Sub}) : K1, \text{cbasic}(\text{Sub}) : K2]. \end{aligned}$$

In this example we can see two *m-predicates* cassb and cbasic were the respective multiplicity variable are *Qty* (first rule) and K (second rule). The first rule transforms the quantity information *Qty* stored in $\text{assbl}(\text{Part}, \text{Sub}, \text{Qty})$, in a multiplicity of predicate cassb , e.g. if there exists the fact $\text{assbl}(p1, p2, 2)$ the first rule derives two multiplicity facts $\text{cassb}(p1, p2) : 1$ and $\text{cassb}(p1, p2) : 2$, and in the interpretation are stored only the fact $\text{cassb}(p1, p2) : 2$. The last two rules with *m-predicate* cbasic are used to compute the number of basic elements of a particular complex object. In the last rule the FS-goal $K : [\text{cassb}(\text{Part}, \text{Sub}) : K1, \text{cbasic}(\text{Sub}) : K2]$ computes the number of basic element. Consider the following example where we have two basic parts $\text{cbasic}(p1) : 1$ and $\text{cbasic}(p2) : 1$, and the complex object p is formed by two basic object of $p1$ and one basic object of $p2$ ($\text{cassb}(p, p1) : 2$ and $\text{cassb}(p, p2) : 1$). Therefore, we have the following three ground instances that satisfy $\text{cassb}(\text{Part}, \text{Sub}) : K1, \text{cbasic}(\text{Sub}) : K2$ conjunction:

$$\begin{aligned} \text{cassb}(p, p1) : 2, \text{cbasic}(p1) : 1 \\ \text{cassb}(p, p1) : 1, \text{cbasic}(p1) : 1 \\ \text{cassb}(p, p2) : 1, \text{cbasic}(p2) : 1 \end{aligned}$$

The derived result is $\text{cbasic}(p1) : 3$.

Moreover, as it possible to see in the last rule of the previous example, we can give recursive definitions that use the FS-goal, this because the FS-goal is monotone. We prove, by an elegant rewriting of the FS-goal by using lists, that the immediate consequence operator of Datalog^{FS} is monotone. We show how this simple extension is able to express more queries in the context of social networks, and work with Markov chains. More details are given in Chapter 5.

1.4 Contributions of the thesis

In this section we summarized our contributions in each of the contexts above discussed.

Inverse Frequent set Mining

- We define IFM_S problem where the itemset not in S has support equal to zero, prove that its complexity is NP-complete and provides an heuristic level-wise algorithm for its resolutions.
- We provide a *pregeneration* method to extract new itemsets not contained in S .
- We give some experimental results showing that the heuristics algorithm for IFM_S problem combined with the *pregeneration* is a good approach to IFM problem.
- We define the $\text{IFM}_{\sigma'}$ problem where the itemset not directly specified in input instance are constrained to be infrequent i.e. they have a support less or equal to a specified unique threshold. Its complexity is NEXP-complete.
- We find a parameter K relatives to the number of items. Such parameter allows to define a parametrization of $\text{IFM}_{\sigma'}$, called $\kappa\text{-IFM}_{\sigma'}$, that can be reduced to original IFM problem, thus they have the same complexity NP-hard and PSPACE.
- We show that the $\text{IFM}_{\sigma'}$ can be formulated as an integer linear problem with huge number of variables and constraints, in general both exponential in the size of the input. At the same time we show that the parametric versions can be solve as an integer linear problem only with huge number of variable.
- In order to concretely solve $\kappa\text{-IFM}_{\sigma'}$ we relax the integer constraints over the number of duplicates and used the column generation method that is an efficient way to solve particular linear problem with an huge number of variable.
- Since to use the column generation method it is necessary solve its specific pricing problem, we study its complexity (NP-complete) and proposed a formulation of ILP and a polynomial heuristic.
- We show by experimental results that the approach for $\text{IFM}_{\sigma'}$ is efficient and effective and the relaxed integer approximation did not compromise the quality of results. Moreover, despite the problem presents a strong complexity, the real life instances considered, present a very small value of k parameter, and therefore they are tractable with our approach.

Count constraint and inverse OLAP problem

- We provide a new language to define count constraints in relational schema whose special cases are well known relational integrity constraints.
- Similarly to IFM problem we define the inverse OLAP problem: given a set of attribute domains and a set of count constraints verify if there exists (and eventually find) a relational database over the given domains that satisfies the constraints.
- We prove that inverse OLAP is NEXP-complete under data-complexity, program-complexity and combined complexity.
- We individuate many significative sub-problems of inverse OLAP.
- We show how the Inverse OLAP problem can be a step toward data exchange with aggregate constraints.

Datalog with frequency support goals

- We provide an extension of Datalog, called Datalog^{FS} , that enables us to query and reason about the number of distinct occurrences satisfying given goals, or conjunction of goals, in rules.
- We prove by an elegant rewriting of frequency support goal that the immediate consequence operator of Datalog^{FS} is monotone and continuous. This allow us to also write aggregate recursive query in our language.
- We extend the traditional techniques for classical Datalog programs optimizations as differential fixpoint and magic set to Datalog^{FS} .
- We introduce an optimization technique that compute in efficient way Datalog^{FS} program with the count variable is in the head.
- We show as Datalog^{FS} extends the application range to support page-rank and social-network queries

1.5 Organization of the thesis

The thesis is organized as follows:

In Chapter 2 we describe the IFM_S problem with its complexity analysis and a heuristic solution. Moreover we provide the pregeneration methods and show the experimental results.

In Chapter 3 we give a formalization of IFM_{σ'} problem, its parametrization $\kappa\text{-IFM}_{\sigma'}$ and the integer linear programming formulation of both. Moreover we provide an approximation of $\kappa\text{-IFM}_{\sigma'}$ based on the relaxation of integer constraints. In addition we proposed a resolution method based on the column generation approach and provide its relative pricing problem. In this Chapter, we also study the complexity of the pricing problem, its integer linear programming formulation and an heuristic algorithm as its solution. Finally we show the experimental results.

In Chapter 4 we provide an extensive language to define count constraints. Next, we give the formulation of inverse OLAP and we analyzed its complexity in depth. Moreover, several examples are used to show the applicability of our results, and

in particular we describe how inverse OLAP is a step toward data exchange with aggregates .

In Chapter 5 we define an extension of Datalog with frequency support goals, called Datalog^{FS}. We provide the syntax and semantic of the new language. Then, many applications about page rank and social networks queries are formulated with our language.

Finally, in Chapter 6 concludes the thesis.

Inverse Frequent Itemset Mining Problem (IFM)

The Inverse Frequent itemset Mining (IFM) is the problem of computing a transaction database D satisfying specified support constraints on a given set S of itemsets, that are typically the frequent ones. Earlier studies focused on investigating computational and approximability properties of this problem, that is NP-hard. In particular, an interesting subproblem formulation is considered where the transaction (itemset) in D are the only itemset in S and minimum and maximum support constraints can be defined on each itemset. Within this setting, an algorithm is proposed that always satisfies the maximum support constraints, but which treats minimum support constraints as soft ones that are enforced as long as it possible. In fact, a thorough experimentation evidences that minimum support constraints are hardly violated in practice, and that such negligible degradation in accuracy (which is unavoidable due to the theoretical intractability of the problem) is compensated by very good scaling performances.

2.1 Introduction

The *inverse frequent set mining* problem is the problem of computing a database on which a given collection of itemsets must be “frequent” [Mie03]. This problem attracted much attention in the recent years, due to its applications in privacy preserving contexts [WW05, WWL05] and in defining generators for benchmark data [RMZ03]. In particular, earlier studies mainly focused on investigating its computational properties, by charting a precise picture of the conditions under which it becomes intractable (see, e.g., [Cal04, Cal08, Mie03]), and by observing that in its general formulation it is NP-hard even if one looks for approximate solutions [WW05].

In this chapter, the inverse frequent set mining problem is reconsidered from a pragmatic point of view instead. Indeed, we concentrate on defining heuristic approaches that are able to efficiently and efficaciously solve the problem in real-world scenarios. In particular, we consider the original formulation of the problem in [Mie03], where the “frequency” of any itemset in the database is measured in terms

of its support, i.e., as the number of the transactions in which it occurs. Note that other approaches to the inverse frequent set mining problem (e.g., [Cal04, Cal08]) considered the actual frequency, i.e., the support divided by the total number of transactions; however, as discussed in [Mie03], supports convey more information than frequencies and hence the perspective of [Mie03] is adopted here. In fact, while keeping this perspective, we investigate a more general setting obtained by relaxing the simplifying assumption in [Mie03] that the size of the output database must be known beforehand, and by furthermore considering a minimum and a maximum support constraint (over each itemset required to occur in the database) in place of a single support value.

2.2 Problem Formalization and Complexity Issues

Let $\mathcal{I} = \{o_1, \dots, o_n\}$ be a finite domain of elements, also called *items*. Any subset $I \subseteq \mathcal{I}$ is called an itemset over \mathcal{I} . The universe of itemsets $\mathbf{U}_{\mathcal{I}}$ is the set of all non-empty itemsets over \mathcal{I} . A database \mathcal{D} over \mathcal{I} is bag of itemsets, each one usually called *transaction*. The number of transactions in \mathcal{D} is denoted by $|\mathcal{D}|$. Given a database \mathcal{D} over \mathcal{I} , for each itemset I over \mathcal{I} ($I \in \mathbf{U}_{\mathcal{I}}$), the *support* of I , denoted by $\sigma^{\mathcal{D}}(I)$, is the number of transactions containing I , and the number of duplicates of I , denoted by $\delta^{\mathcal{D}}(I)$, is the the number of transactions equal to I . We say that I is a *frequent* itemset in \mathcal{D} w.r.t. a given support threshold s if $\sigma^{\mathcal{D}}(I) \geq s$. Observe that supports are also represented in the literature as a percentage w.r.t. the dimension of \mathcal{D} , i.e. by $\sigma^{\mathcal{D}}(I)/|\mathcal{D}|$. Finding all the frequent itemsets in \mathcal{D} is the well-known *frequent itemset mining problem*. The *anti-monotonicity property* holds for supports: given two itemsets I and J with $I \subset J$, $\sigma^{\mathcal{D}}(J) \leq \sigma^{\mathcal{D}}(I)$.

We denote the set of natural numbers by \mathbb{N}_0 that will be used for bound. We also introduce the symbol ∞ to denote an unlimited bound and define $\tilde{\mathbb{N}}_0$ as $\mathbb{N}_0 \cup \{\infty\}$ — we therefore assume that for each $i \in \mathbb{N}_0$, $i < \infty$ holds. Finally, we denote the set of pairs $\{(a, b) : a \in \mathbb{N}_0, b \in \tilde{\mathbb{N}}_0, a \leq b\}$ by $\tilde{\mathbb{N}}^2$. In the chapter, we consider the *inverse frequent itemset mining problem*: Given a set S of itemset, finding a database in which each element of S is frequent. This is formalized below.

Definition 2.1 (IFM Problem).

Let:

1. S be a given set of itemsets over the items in \mathcal{I}
2. $\Gamma_{\sigma} = \{(I, \sigma_{min}^I, \sigma_{max}^I) : I \in S, (\sigma_{min}^I, \sigma_{max}^I) \in \tilde{\mathbb{N}}^2\}$ be a given set of triples assigning a minimum and the maximum support to each itemset in S

Then, the *Inverse Frequent Itemset Mining Problem* on \mathcal{I} , S and Γ_{σ} (short: IFM(\mathcal{I} , S , Γ_{σ})) consists of finding a database \mathcal{D} over \mathcal{I} such that the following condition hold (or eventually state that there is no such a database):

$$\forall I \in S : \sigma_{min}^I \leq \sigma^{\mathcal{D}}(I) \leq \sigma_{max}^I \quad (2.1)$$

□

It is worthwhile noticing that the inverse frequent mining problem was mainly formulated and analyzed in the literature within contexts where the *frequency* of an itemset (i.e. its support $\sigma^{\mathcal{D}}$ divided by the total number of transactions in \mathcal{D}) is considered in place of the support in the problem formulation. When using this perspective, IFM is generally referred to as the FREQSAT problem, and various complexity results are known for it. For instance, it is well-known that FREQSAT is NP-complete (even if the size of each transaction is bounded by some given input parameter or if $\sigma_{min} = \sigma_{max}$) and \mathcal{PP} -hard—hence, intrinsically more complex—if the maximal number of duplicates of any transaction is bounded by some parameter [Cal04].

Considering the support of the itemsets rather than their frequency received considerably less attention instead. In particular, in [Mie03], a slight variation of IFM was studied where the size $|\mathcal{D}|$ of the output database \mathcal{D} is fixed beforehand (short: $\text{IFM}_{|\mathcal{D}|}$). This variation was observed to be NP-hard, even if $\sigma_{min} = \sigma_{max}$. However, the complexity of the main problem IFM (i.e., in absence of such an additional constraint on the size of \mathcal{D}) was not derived in earlier literature. Our first contribution is precisely to complete the picture of the complexity issues arising with the decision version of the inverse frequent mining problem. Indeed, we shall evidence that IFM is computationally intractable too, thereby calling for (heuristic) solution approaches that are efficient in actual scenarios.

Proposition 2.2. IFM is NP-hard, also when $\sigma_{min}^I = \sigma_{max}^I$ for all I in S .

Note that there are several sources of intractability in the formulation of IFM. The first one is that the “structure” of the various transactions to be inserted into \mathcal{D} is not known beforehand. Therefore, when building such database we are uncertain on which kinds of transaction to exploit. Towards devising heuristic approaches for IFM, a natural idea is then to consider a simplification of IFM where for each itemset J in $\mathcal{U}_{\mathcal{I}} \setminus S$ not frequently, $\sigma^{\mathcal{D}}(J) = 0$ or $\delta^{\mathcal{D}}(J) = 0$, i.e. all the possible transactions are taken from the set S of itemsets provided in input.

Definition 2.3 (IFM_S Problem).

Let:

1. S be a given set of itemsets over the items in \mathcal{I}
2. $\Gamma_{\sigma} = \{(I, \sigma_{min}^I, \sigma_{max}^I) : I \in S, (\sigma_{min}^I, \sigma_{max}^I) \in \mathbb{N}^2\}$ be a given set of triples assigning a minimum and the maximum support to each itemset in S

Then, the IFM_S Problem on \mathcal{I} , S and Γ_{σ} (short: $\text{IFM}_S(\mathcal{I}, S, \Gamma_{\sigma})$) consists of finding a database \mathcal{D} over \mathcal{I} such that the following two conditions hold (or eventually state that there is no such a database):

$$\begin{aligned} \forall I \in S : \sigma_{min}^I &\leq \sigma^{\mathcal{D}}(I) \leq \sigma_{max}^I \\ \forall J \in \mathcal{U}_{\mathcal{I}} \setminus S : \delta^{\mathcal{D}}(J) &= 0 \end{aligned}$$

□

Rather surprisingly, we next show that even IFM_S is intractable in general.

Theorem 1 IFM_S is NP-complete in general whereas it is feasible in polynomial time if $\sigma_{min}^I = \sigma_{max}^I$ for all I in S .

Proof.

Let us first consider the general case. To prove that IFM_S is in NP, we observe that the size of the output database \mathcal{D} is certainly bounded by $|S|$ and by the largest support required in the specification. Hence, in polynomial time a non-deterministic Turing machine may first guess such database (basically, the support for each itemset I in S) and then verify whether it satisfies the constraints σ_{min}^i and σ_{max}^i , by simply computing the support $\sigma^{\mathcal{D}}(I_i)$ on \mathcal{D} .

We now prove that IFM_S is NP-hard by exhibiting a reduction from the *graph 3-colorability problem* of deciding whether, given a graph $G = (V, E)$, there is a 3-coloring $c : V \rightarrow \{r, g, b\}$ such that $c(i) \neq c(j)$ for each pair of edges $(i, j) \in E$.

Based on the input graph $G = (V, E)$, we construct an instance of the $\text{IFM}_S(\mathcal{I}, \Gamma_S)$ problem such that: the set \mathcal{I} of items is $\{r, g, b, l_1, l_2, l_3\} \cup \{v_x | x \in V\} \cup \{e_{z,y} | (z, y) \in E\}$, where conceptually the item r, g, b are the colors in G , l_1, l_2, l_3 are labels implementing an encoding of the three colors, v_x is an item for each node in G and $e_{z,y}$ is an item for each edge in G . The encoding of colors by label is such that any two colors share exactly one label; in the proof we shall use the encoding $r = \{l_1, l_2\}$, $g = \{l_1, l_3\}$ and $b = \{l_2, l_3\}$.

The set Γ_S contains two groups of constraints:

Group (I): these constraints are repeated for each node $x \in V$ and enforces that x must be colored with exactly one color. There are 7 itemsets associated to x organized on 3 levels:

- 3 itemsets at the highest level 2: there is an itemset for each possible color c for x , containing the items corresponding to the node x , to all the arcs leaving x , to the color c and to the encoding of the color — the support for such itemsets can be either 0 or 1;
- 3 itemsets at level 1: there is an itemset for each of the 3 encoding labels, containing the item corresponding to the label and the item corresponding to the node x – the support must be exactly 1;
- 1 itemset at level 0 containing the item corresponding to the node x – its support must be exactly 2;

We explicit the constraints below:

- $(I_{x,r}, 0, 1)$, $(I_{x,g}, 0, 1)$, $(I_{x,b}, 0, 1)$, where $I_{x,r} = \{v_x, r, l_1, l_2\} \cup I$, $I_{x,g} = \{v_x, g, l_1, l_3\} \cup I$, $I_{x,b} = \{v_x, b, l_2, l_3\} \cup I$, and $I = \{e_{x,y} | (x, y) \in E\}$;
- $(\{l_1, v_x\}, 1, 1)$, $(\{l_2, v_x\}, 1, 1)$ and $(\{l_3, v_x\}, 1, 1)$;
- $(\{v_x\}, 2, 2)$.

Because of the support constraints for the itemsets at level 1, $\{l_1, v_x\}$ cannot occur as transaction in \mathcal{D} as it inherits support from those itemsets; in addition, as the support of $\{l_1, v_x\}$ is 2 and there are 3 itemsets at level 1 with obligatory support 1, exactly one of the itemsets at level 1 must occur as transaction, whereas the other two inherit support from a same itemset at level 2. It turns out that exactly one itemset at level 2

can occur as transaction whereas all others must have support 0 - the itemset selected as transaction will then fix the unique color for the node x .

Group (II): these constraints are repeated for each edge $(x, y) \in E$ and enforce that two end nodes of the edge have different color. There are 3 itemsets, one for each possible color; the constraints are:

- $(\{r, e_{z,y}\}, 0, 1), (\{g, e_{z,y}\}, 0, 1), (\{b, e_{z,y}\}, 0, 1)$.

The above itemsets inherit support from two itemsets at level 2: one for x and the other for y . The constraints of Group (II) enforces that the two itemsets at level 2 cannot be of the same color, thus any two adjacent nodes cannot share the same color. It follows that the existence of a solution to IFM_S witnesses the fact that the graph G admits a 3-coloring; on the other hand, if IFM_S has no solution then the graph cannot be 3-colored. Hence, IFM_S is NP-hard in general.

To conclude the proof, we observe that in the special case where $\sigma_{min}^I = \sigma_{max}^I$ holds for all itemsets I in S , one may encode the solutions of IFM_S in terms of a system of linear equations over $|S|$ variables and constraints. The result then follows, since it is well-known that deciding whether a system of linear equations admits a solution is feasible in polynomial time — note that in this case an integer number solution is also required. \square

2.3 A Level-Wise Solution Approach for IFM_S

In order to devise an efficient and effective algorithm for IFM , we find convenient to firstly tackle the special case where the output database must be constructed by using as transactions the input itemsets only (cf. IFM_S problem), and then to build the general algorithm for IFM on top of the solution approach for IFM_S .

In fact, given that it is not possible to efficiently enforce both the minimum and the maximum support on each itemset in S (cf. Theorem 1), we shall propose in this section an approach to face IFM_S which takes care for all I in S of the constraint over σ_{max}^I only, while treating the constraint over σ_{min}^I as a soft constraint that must be satisfied as long as it possible. Here, it is worthwhile anticipating that a thorough experimental activity conducted on our solution approach evidenced that the output of our technique will hardly violates for all I in S the constraint on σ_{min}^I (though they are treated as soft ones).

2.3.1 Description of the Algorithm

Recall that $\text{IFM}_S(\mathcal{I}, S, \Gamma_\sigma)$ amounts to finding a database \mathcal{D} that is built over the itemsets in S (each one possibly occurring with multiple repetitions) and where constraints associated to the set Γ_σ are satisfied. Thus, a natural approach to face IFM_S is to iterate over the elements in S and decide how many copies have to be added in the output database for each of them. However, various strategies can be used to process the elements in S and to decide about the number of copies to be added for each of them. Our strategy is based on two key ideas.

Input: Set S of itemsets over \mathcal{I} , set Γ_σ .
Output: Database \mathcal{D} .
Method:

- 1 let S_1, \dots, S_k be the succession as in Section 2.3.1;
- 2 $\mathcal{D} := \emptyset$;
- 3 **for each** “level” $i : 1$ to k **do**
- 4 optimizeSelection(i);
- 5 **for each** itemset $I \in S_i$ such that $\Delta\sigma_{min}^{\mathcal{D}}(I) > 0$ **do**
- 6 insert $\Delta\sigma_{min}^{\mathcal{D}}(I)$ copies of I in \mathcal{D} ;
- 7 **end for**
- 8 **return** \mathcal{D} ;

Procedure optimizeSelection(i : level in $\{1, \dots, k\}$):

- P1 $B := bestCandidates(S_i, \mathcal{D})$;
- P2 **while** $B \neq \emptyset$ **do**
- P3 select an itemset $I \in B$;
- P4 insert $inc(I, S_i, \mathcal{D})$ copies of I in \mathcal{D} ;
- P5 $B := bestCandidates(S_i, \mathcal{D})$;
- P6 **end while**

Fig. 2.1. Solution Algorithm for IFM $_S$.

Firstly, we propose to process the itemsets of S that are candidates for being added to the output database \mathcal{D} by means of a level-wise approach, where larger (w.r.t. set containment) itemsets are processed first. Formally, let S_1, S_2, \dots, S_k be a succession of subsets of S such that: (i) $S_i \cap S_j = \emptyset$ for each $i \neq j$; (ii) $\bigcup_{i=1}^k S_i = S$; and, (iii) for each pair of subsets S_i and S_j with $i < j$, and for each itemset $J \in S_j$, there is an itemset $I \in S_i$ s.t. $I \supset J$ and there is no itemset $I' \in S_i$ s.t. $I' \subseteq J$. Note that the succession S_1, S_2, \dots, S_k can efficiently be computed from S , and that by processing itemsets according to their order of occurrence in the succession, we can enforce that each itemset is processed prior to its subsets.

Secondly, whenever processing the i -th element S_i of the above succession and for each itemset $I \in S_i$, we propose to add in \mathcal{D} the minimum possible number of copies of I that suffices to satisfy σ_{min}^I and that do not lead to violate the maximum support constraints on the subsets of I , which have in fact to be still processed. Formally, let $\Delta^{\mathcal{D}}(I) = \min_{J \in S | J \subset I} (\sigma_{max}^J - \sigma^{\mathcal{D}}(J))$. Then, such number of copies is given by the expression $\Delta\sigma_{min}^{\mathcal{D}}(I) = \min(\sigma_{min}^I - \sigma^{\mathcal{D}}(I), \Delta^{\mathcal{D}}(I))$. Note that one may obtain $\Delta\sigma_{min}^{\mathcal{D}}(I) \leq 0$, thereby implying that the algorithm fails in providing the minimum required support for I .

An algorithm implementing these ideas is shown in Figure 2.1. The input of the algorithm is a set of itemsets S and a set Γ_σ denoting the support constraints associated with such itemsets. The output is a database with itemsets as transactions, which is meant as a heuristic solution to IFM $_S(\mathcal{I}, S, \Gamma_\sigma)$.

The algorithm starts by setting the database \mathcal{D} to the empty set, and then applies the level-wise exploration of the itemsets in S in order to add elements into \mathcal{D} —for the moment let us get rid of step 4 that implements an optimization discussed in Section 2.3.2. In fact, given that each update on \mathcal{D} preserves the maximum support constraint on all the subsets of the processed itemset and given that itemsets are processed according to their set inclusion, it is immediate to check that the resulting database \mathcal{D} is such that for each item $I \in S$, $\sigma^{\mathcal{D}}(I) \leq \sigma_{max}^I$ holds. However, we have no theoretical guarantee on the fact that the minimum support constraint is satisfied over each itemset. In an extreme case, no copy of some itemset might be added to \mathcal{D} even though a certain number of them are required.

As for the running time, note that the dominant operation in step 6 is repeated $|S|$ times, i.e., once for each itemset in S . Moreover, computing $\Delta\sigma_{min}^{\mathcal{D}}(I)$ requires iterating over all the subsets of I , which are $|S|$ at most. In total, $O(|S|^2 \times |\mathcal{I}|)$ is a bound on the running time of the algorithm in Figure 2.1, where the $|\mathcal{I}|$ factor accounts for the cost of manipulating (e.g., comparing) itemsets composed by $|\mathcal{I}|$ items at most after an arbitrary ordering is fixed on them.

2.3.2 Optimization Issues

In the algorithmic scheme we have discussed, at each level i of the search, itemsets from S_i only are added into \mathcal{D} . In practice this might be too restrictive. Indeed, we might think of enforcing the support of each itemset $I \in S_i$ by adding an itemset $J \supset I$ contained in some set S_j with $j < i$, rather than by directly adding I . In fact, note that such an itemset S_j is guaranteed to exist (except for $i = 1$), by construction of the succession S_1, \dots, S_k .

This optimization founds on the idea that enforcing the support of I by including copies of one of its supersets has the side-effect of incrementing the support of other itemsets included in J and, hence, belonging to same level subsequent to S_j . This is very relevant in our approach, given that this effect goes in the direction of amplifying the chances of ending up with a database satisfying the minimum support constraints over all itemsets, which is a critical issue as we discussed above. In practice, to implement this strategy, the algorithm in Figure 2.1 accounts for an optimization step that is performed before that itemsets at the current level S_i are analyzed. This optimization is next discussed in detail.

Let $\Delta\sigma_{max}^{\mathcal{D}}(I) = \min(\sigma_{max}^I - \sigma^{\mathcal{D}}(I), \Delta^{\mathcal{D}}(I))$ be the maximum number of copies of I that can be added to \mathcal{D} while still satisfying σ_{max}^I and while not violating the maximum support constraints on the subsets of I . For any itemset $I \in S_j$ and for any element S_i with $i > j$, define then $inc(I, S_i, \mathcal{D})$ as the value:

$$\min \left(\Delta\sigma_{max}^{\mathcal{D}}(I), \min_{\substack{I' \in S_i \wedge I' \subset I \wedge \\ \sigma_{min}(I') > \sigma^{\mathcal{D}}(I')}} (\sigma_{min}^{I'} - \sigma^{\mathcal{D}}(I')) \right).$$

Intuitively, this is the maximum increment allowed on the support of I computed by also considering as a bound the minimum support which suffices to satisfy the

constraints $\sigma_{min}^{I_s}$ on any of its subsets I_s that have still to be satisfied (i.e., for which $\sigma_{min}^{I_s} > \sigma^{\mathcal{D}}(I_s)$ currently holds). In practice, we do want to increment the support of I by affecting as less as possible the support of its subsets. Based on the increment values computed at the level associated with S_i , we want to compute the set of all the itemsets in some level below S_i that leads to reduce as much as possible the number of itemsets to be added to \mathcal{D} .

To this end, define first $gainSet(I, S, J, \mathcal{D})$ as the itemsets whose minimum support is not yet satisfied and that are subsets of I and supersets of J , i.e., $gainSet(I, S, J, \mathcal{D}) = \{I' \in S \mid I' \subset I \wedge I' \supset J \wedge \sigma_{min}^{I'} < \sigma^{\mathcal{D}}(I')\}$. Define then $gain(I, S_i, J, \mathcal{D})$ as the value:

$$\frac{inc(I, S_i, \mathcal{D}) * (|gainSet(I, S, J, \mathcal{D})| - 1)}{\Delta\sigma_{max}^{\mathcal{D}}(I)}.$$

Intuitively, this is a normalized value that is meant to denote the advantage of adding $inc(I, S_i, \mathcal{D})$ copies of I to \mathcal{D} w.r.t. an itemset J that have still to be processed. By averaging this gain over all the itemsets that have to be processed, we define the value $avgGain(I, S_i, \mathcal{D})$ as:

$$\frac{\sum_{J \in \bigcup_{j=i+1}^k S_j \wedge J \supset I} gain(I, S_i, J, \mathcal{D})}{|\bigcup_{j=i+1}^k S_j|}.$$

Finally, let $bestCandidates(S_i, \mathcal{D})$ be the set of all itemsets in $\bigcup_{j=1}^{i-1} S_j$ on which the maximum value (not equals to zero) of the average gain is achieved. These itemset are those that we consider the most promising for being added to \mathcal{D} . These itemsets are computed in step P1 of the optimization procedure and updated in P5, after that $inc(I, S_i, \mathcal{D})$ copies of an itemset I (arbitrarily picked from $bestCandidates(S_i, \mathcal{D})$) have been actually added to \mathcal{D} .

As for the running time, note that while without optimization we need to iterate only over the sets in $|S|$ (as to compute $\Delta\sigma_{min}^{\mathcal{D}}(I)$), we now need to compute the set $bestCandidates(S_i, \mathcal{D})$ which is feasible in $O(|S^2|)$. In total, the complexity is now $O(|S^3| \times |\mathcal{I}|)$.

2.3.3 An Example of computation

We conclude the description of the algorithm in Figure 2.1 by illustrating an example of computation over an IFM_S problem defined over the set $\mathcal{I} = \{a, b, c, d\}$ of items, and such that: $S = \{I_1, \dots, I_9\}$ where $I_1 = \{a, c, d\}$, $I_2 = \{a, b, c\}$, $I_3 = \{c, d\}$, $I_4 = \{c, a\}$, $I_5 = \{b, c\}$, $I_6 = \{d\}$, $I_7 = \{a\}$, $I_8 = \{c\}$, and $I_9 = \{b\}$; and where the functions σ_{min} and σ_{max} are those defined in the following table:

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9
σ_{min}	1	1	1	4	5	1	4	5	5
σ_{max}	4	5	4	5	6	4	5	6	6

Fig. 2.2. Example in Section 2.3.3: Input Itemsets (top). Database Generation (bottom).

Note that itemsets in S can be arranged into the three levels S_1 , S_2 , and S_3 , as for they are graphically depicted in the topmost part of Figure 2.2. Thus, at the first iteration of the algorithm, all the elements in S_1 are processed. No optimization can be performed on them since this is the first level of the hierarchy, and hence the algorithm performs steps 5 and 6 by updating the database \mathcal{D} (initially empty) by adding one copy of I_1 and one copy of I_2 . In fact, note that $\Delta\sigma_{min}^{\mathcal{D}}(I_1) = \Delta\sigma_{min}^{\mathcal{D}}(I_2) = 1$.

In the second iteration, the algorithm processes the second level S_2 . This time, the optimization procedure can be applied to S_1 . Note that we have: $gain(I_1, S_2, I_6, \mathcal{D}) = gain(I_1, S_2, I_7, \mathcal{D}) = gain(I_2, S_2, I_7, \mathcal{D}) = \frac{3*(1-1)}{3} = 0$, $gain(I_1, S_2, I_8, \mathcal{D}) = gain(I_2, S_2, I_8, \mathcal{D}) = \frac{3*(2-1)}{4} = 0.75$, and finally $gain(I_2, S_2, I_9, \mathcal{D}) = \frac{3*(1-1)}{5} = 0$.

Thus, $avgGain(I_1, S_2, S_3, \mathcal{D})$ coincides with the value:

$$\frac{gain(I_1, S_2, I_6, \mathcal{D}) + gain(I_1, S_2, I_7, \mathcal{D}) + gain(I_1, S_2, I_8, \mathcal{D})}{4} = 0,$$

whereas $avgGain(I_2, S_2, S_3, \mathcal{D})$ is the value:

$$\frac{gain(I_2, S_2, I_7, \mathcal{D}) + gain(I_2, S_2, I_8, \mathcal{D}) + gain(I_2, S_2, I_9, \mathcal{D})}{4} = 0.18.$$

It follows that, at the end of the second iteration, the algorithm updates \mathcal{D} by adding $inc(I_2, S_2, \mathcal{D}) = 3$ copies of I_2 into \mathcal{D} . The reader may now check that the subsequent iteration ends up by adding one copy of I_5 . After this update, the database \mathcal{D} turns out to satisfy all the constraints (see the bottom part of Figure 2.2).

2.4 Solving the General Case of IFM

Now that a solution approach for IFM_S has been described, we can move to discussing a solution approach for the more general IFM problem. In fact, it is worthwhile observing that the algorithm in Figure 2.1 can already be seen as a heuristic approach to face IFM. Indeed, in addition to the specific heuristic used to deal with the support constraints, this algorithm heuristically solves IFM by restricting the set of all the possible transactions to those in S .

In practice, focusing on the set S might be too restrictive to solve IFM. Thus, we propose to enlarge the original set S by including various novel itemsets (built from those in S), whose exploitation is envisaged to be beneficial for improving the effectiveness of the algorithm in Figure 2.1.

The approach, illustrated in Figure 2.3, constructs a novel set S' to be used as input for the algorithm in Figure 2.1 by merging k itemsets at most from S . The merging function is carried out recursively, by picking an itemset at time from a set of candidates itemsets (S_{cand}).

In particular, it is worthwhile observing that itemsets are merged together if and only if their intersection is not empty (which motivates the initialization in step 2 and the check in F5). Indeed, this is in line with the approach discussed in Section 2.3.2, where the gain of two itemsets having no subset in common is equals to zero. In addition, note that the merging process is restricted to maximal itemsets only, i.e., to those which are included in the first level S_1 .

We conclude the section by noticing that as for the support constraints, for each itemset $I \in S'$, we set $\sigma'_{min}(I)$ as the value:

$$\begin{cases} \sigma_{min}^I & I \in S \\ \max(0, \max(\sigma'_{min}(Y) | Y \in S' \wedge I \subset Y)) & I \notin S \end{cases}$$

Input: Set S of itemset, integer k .

Output: Set S' of itemset.

Method:

```

1 let  $S_1, \dots, S_k$  be the succession as in Section 2.3.1;
2  $S_{cand} = \{I \in S_1 | \exists Y \in S_{i>1} : Y \subset I\}$ ;
3  $S' = \{\emptyset\}$ ;
4 for each itemset  $I \in S_{cand}$  do
5    $S_{cand} = S_{cand} - \{I\}$ ;
6    $S_{fusion} = \{I\}$ ;
7    $S' = S' \cup \text{recursGen}(S_{fusion}, S_{cand}, k - 1)$ ;
8 end for
9 return  $S'$ ;

```

Function $\text{recursGen}(S_{fusion}, S_{cand}$: set of itemsets, k : integer): set of itemsets;

```

F1 if  $k = 0$  then return  $\{\emptyset\}$ ;
F2 let  $S^* = \{\emptyset\}$ ;
F3 for each itemset  $I \in S_{cand}$  do
F4    $S_{cand} = S_{cand} - \{I\}$ ;
F5   if  $\bigcap_{Y \in S_{fusion}} Y \cap I \neq \emptyset$  then
F6      $S^* = S^* \cup \{\bigcup_{Y \in S_{fusion}} Y \cup I\} \cup$ 
        $\cup \text{recursGen}(S_{fusion} \cup \{I\}, S_{cand}, k - 1)$ ;
F7 end for
F8 return  $S^*$ ;

```

Fig. 2.3. Preprocessing for IFM.

and we set $\sigma'_{max}(I)$ as the value:

$$\begin{cases} \sigma_{max}(I) & I \in S \\ \min(st, \min(\sigma'_{max}(Y) | Y \in S' \wedge I \supset Y)) & I \notin S \end{cases}$$

where st is a user bound on the support of the new itemset.

Given these novel input specifications, the algorithm for $\text{IFM}_{S'}$ is then applied and used as a heuristic approach to solve IFM on S . Eventually, note that for any fixed natural number k , the above preprocessing step is not an overhead for the running time of the algorithm in Figure 2.1.

2.5 Computational Result

The above solution approach for IFM has been implemented, and a thorough experiential activity has been conducted to assess its efficiency and effectiveness. Details on this activity are discussed in the remaining of the section.

2.5.1 Test instance

Experimentation is carried out over three distinct datasets [ZKM01], which have been often used as reference benchmarks for frequent itemsets discovery algorithms: The artificial dataset T10I4D100K, and the two real datasets BMS-WebView-1 and BMS-WebView-2. These latter databases contain clickstream data from two e-commerce web sites (each transaction represents a web session and each item in a page viewed in that session). Summary information on these three datasets (in particular, the total number of transactions, the maximum transaction size, and the average transaction size, the number of distinct items, and the average frequency of an item) are illustrated in Table 3.4.

DB	Ntrans	Max. Trans. size	Avg. Trans. size	# items	Avg. Frequency Item
BMS-WebView-1	59602	267	2.5	497	0.0050
BMS-WebView-2	77512	161	4.6	3340	0.0013
T10I4D100K	100000	300	10	870	0.0116

Fig. 2.4. Characteristics of used data set.

For each of the above databases, the idea of the experimentation is to firstly extract the set S of the itemsets that are actually frequent (by standard itemsets discovery algorithms). For each itemset $I \in S$ whose frequency is σ , we define $\sigma_{min}^I = \sigma - \alpha * \sigma$ and $\sigma_{max} = \sigma + \alpha * \sigma$, where α is a normalized real number (that will be varied in the experimentations). Then, the whole set S together with the support constraints constructed as above will be supplied as input to our algorithm for the IFM problem.

To evaluate the scalability of the algorithm we shall just refer to its running time. Instead, to evaluate its effectiveness in solving IFM, we shall take care of the following *relative error* index:

$$er(\%) = \frac{1}{|S|} * \sum_{I \in S: \sigma^D(I) < \sigma_{min}^I} \frac{\sigma_{min}^I - \sigma^D(I)}{\sigma_{min}^I}$$

It is worthwhile observing that the above index accounts for how many itemsets from S occur in the output database with a support that is below the required constraint. In fact, recall from Section 2.3.1 that the maximum support constraint is always satisfied with our approach. Clearly enough, values of $er(\%)$ close to 0 are desirable.

All the results discussed below have been obtained by experimenting with an Intel dual core with 1.8GB memory, running windows XP Professional.

2.5.2 Results

A first series of experiment was aimed at assessing the scalability of our approach w.r.t. some key input parameters. In particular, we considered the real datasets BMS-WebView1 and BMS-WebView2 by varying the parameter α (from 0 to 0.3). Execution times are reported in Figures 2.5 and 2.6, for increasingly larger support and size of the set S . Note that execution times are not affected by the size of the interval over the support constraints.

In a second series of experiments, we assessed the effectiveness of the approach by computing the relative index error over the three datasets, by varying the support and the α parameter. Results are reported in Figure 2.7. The figures evidence that the error rate is below 2%

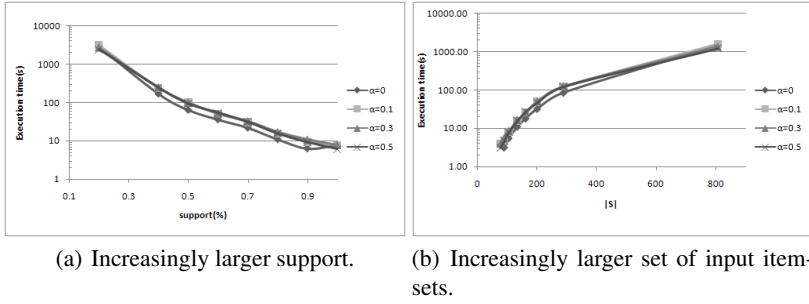


Fig. 2.5. Execution Times on BMS-WebView1 (for different α values).

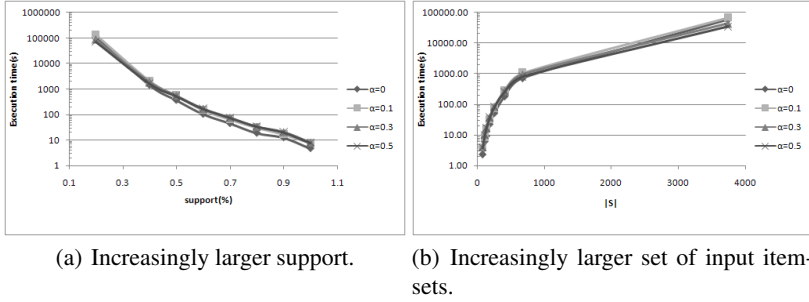


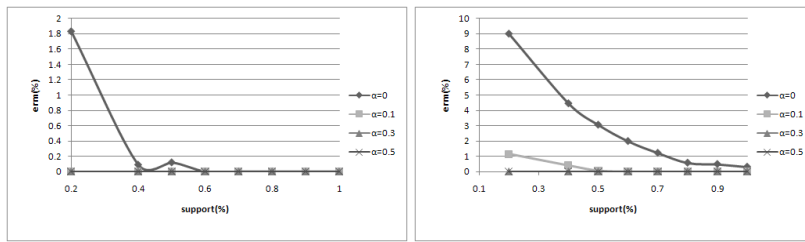
Fig. 2.6. Execution Times on BMS-WebView2 (for different α values).

for BMS-View-1 and T10I4D100K dataset, while is below 10% for BMS-View-2. It comes with no surprise that accuracy improves by increasing α . Moreover, it is relevant to notice that the error is always equals to 0 when $\alpha > 0.3$, i.e., when a sufficiently large support constraint window is defined.

In a further set of experiments, we considered the experimentation perspective discussed in [WWWL05]. There, it is argued that the effectiveness of inverse frequent itemsets mining algorithms has to be assessed (1) by comparing the itemsets that can be actually rediscovered on the syntectic database with those occurring in the original one, and (2) by comparing the performances of a mining algorithm (e.g., Apriori) over the syntectic and the original dataset.

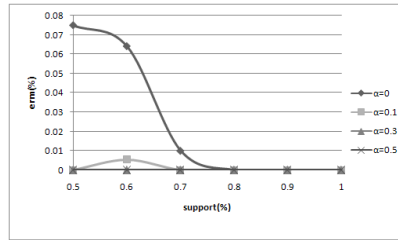
In order to deal with (1) above, we used the *Jaccard*, *Dice*, and *Overlap* [WWWL05] indices to compare similarities between original frequent itemsets and those occurring in the syntectic output database. Results on the real input datasets are reported in Figure 2.8. The figure evidences that very high accuracy measures are obtained, and that support threshold values are greater or equal to the support threshold used for data generation.

Finally, as for (2), we report in Figure 2.9 the difference between execution times of Apriori when running on the original dataset and when running on the syntetic one (build on T10I4D100K by using the three supports values: 0.5, 0.6 and 0.7). Note that the lower is the support used in the generation of dataset, the smaller is the difference of performances.



(a) BMS-WebView1.

(b) BMS-WebView2.



(c) T10I4D100K.

Fig. 2.7. Accuracy of the Approach.

BMS-View1	σ	Jaccard	Dice	Overlap	BMS-View2	σ	Jaccard	Dice	Overlap
support=417	417	1	1	1	support=543	543	0,968	0,984	1
	477	1	1	1		620	0,978	0,989	1
	536	1	1	1		698	0,965	0,982	1
	596	1	1	1		775	0,951	0,975	1

(a) Accuracy measured on *BMS* – View – 1.

(b) Accuracy measured on *BMS* – View – 2.

Fig. 2.8. Accuracy Measured on Generated data.

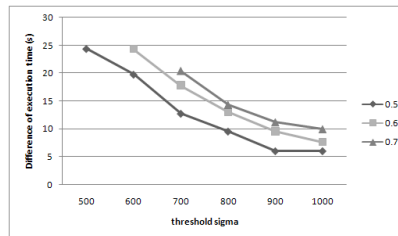


Fig. 2.9. Difference of Execution Times with Apriori (Generated VS Original Data).

A New Generalization of IFM

The Inverse Frequent itemset Mining problem is the problem of computing a transaction database D satisfying a given set S of support constraints on some itemsets that are typically the frequent ones. Earlier studies focused on investigating computational and approximability properties of this problem. However, this classical formulation does not enforce any constraint on the other itemsets, and permits that D contains additional (and, perhaps, unsuspected or even undesired) frequent itemsets. A possibility for removing this anomaly is to introduce a more general formulation of the IFM problem in which itemsets that do not belong to S might be explicitly constrained (e.g., to admit "non-frequent" transactions provided their supports are below some given threshold). Despite this formulation solve the anomaly, its complexity is NEXP-complete. In this chapter we investigate this formulation, define a parametrization whose complexity is PSPACE, show how it can be encoded as an integer linear program with a huge number of variables (in general exponential w.r.t. the constraints), and consider an approximation based on relax integer constraints that turns NP-complete. In order to solve such approximation we use column generation technique that is a method designed to solve optimization problems with a huge number of variables. In our context, it requires at each step the solution of an auxiliary integer linear program, which we prove being still NP-complete. Thus, a constructive heuristic for this auxiliary problem has also been defined, which enjoys very good scaling, thereby paving the way for its application over real-life scenarios.

3.1 Introduction

Transaction databases are databases where each tuple, called *transaction*, is defined as a subset of an underlying fixed set of *items* \mathcal{I} . As an example, in market data, a transaction corresponds to one purchase event and each item is a salable product; in document indexing and search engine applications, a corpus of document can be represented as a transaction database whose transactions are one-to-one associated with the documents, each one viewed as the set of the words occurring in it; in web mining applications, a collection of web pages is yet another example of transaction database, where the items are the links occurring in the various pages and where, for each web page in the collection, there is one transaction containing its outgoing (or alternatively incoming) links.

A popular mining task over transaction databases is to single out the set of the *frequent itemsets*, i.e., all the subsets of \mathcal{I} (called *itemsets*) which are contained in a significant fraction

(user-specified as a minimum support threshold) of the given transactions [WHH00]. This problem attracted relevant research efforts in recent years, and several solution approaches and generalizations have indeed been discussed in the literature. In some cases, however, the perspective of the frequent itemset mining problem is naturally inverted; that is, we might be given in advance a set of itemsets and our goal would then be to decide whether there is a transaction database over which these itemsets are actually frequent (and, of course, compute the database whenever the answer is positive). This problem, called *inverse frequent itemset mining* problem (IFM for short), has been introduced in the context of defining generators for benchmarks of mining algorithms [Mie03], and has been subsequently reconsidered in privacy preserving contexts [WW05, WWL05], where the goal is to publish some data while avoiding disclosing sensitive or private knowledge. A more extensive discussion on the possible applications of IFM is reported in Section 3.6.

From a technical viewpoint, earlier studies on the inverse frequent itemset mining problem investigated its computational properties, by charting a precise picture of the conditions under which it becomes intractable (see, e.g., [Cal04, Cal08, Mie03]), and by observing that the problem is NP-hard even if one looks for approximate solutions [WW05]. In this chapter, the inverse frequent set mining problem is considered from a pragmatic point of view even though our approach is driven by novel theoretical results. We concentrate on studying heuristic solution approaches that are able to scale over large instances, by contextualizing them within the basic inverse frequent itemset mining setting of [Mie03], where the “frequency” of any itemset in the database is measured in terms of its support, i.e., as the number of the transactions in which it occurs¹.

Given a set S of itemsets and support constraints on them as input, our goal is to solve the search problem by providing an approximated solution (i.e., a transaction database) for which the supports on the itemsets in S are satisfied and whose size is polynomially bounded by the input. We assume that, in place of a single support value, each itemset $I \in S$ comes with a range of admissible supports, so that the emergence of unsatisfiable instances can be dramatically decreased by properly enlarging the ranges of tolerance over each itemset. This version of the problem introduces more flexibility and is not at all a limitation as the two range interval extremes may coincide.

We point out that we introduce an important extension to the original framework of [Mie03] pertaining to the role played by the itemsets not in S (itemsets in S' for short). In the current literature, the question of imposing some generic support restriction on itemsets in S' has largely been ignored. Classical approaches take a “liberal” position that there are no constraints on the itemsets in S' . Therefore, some of them could eventually result to be highly supported in the solution database and the designer could be eventually disappointed in discovering such unexpected patterns, that would even be undesired when s/he has listed all the frequent ones in S .

To avoid any undesired behavior of the itemsets non in S , some approaches (see, e.g., [GSS09a, GTTY06]) drastically impose them not to occur as transactions in the generated database. However, this assumption is too restrictive as these approaches are often unable to find any solution, whereas admitting a limited number of transactions built with itemsets in S could make the instance solvable. Moreover, only focusing on the itemsets belonging to S

¹ Note that other approaches to the inverse frequent set mining problem (e.g., [Cal04, Cal08]) considered the actual frequency, i.e., the support divided by the total number of transactions; however, as discussed in [Mie03], supports convey more information than frequencies and hence the perspective of [Mie03] is adopted here.

does not provide any benefit from a computation viewpoint, as the problem remains NP-hard [GSS09a].

Our framework explicitly considers an extra input parameter which is meant to define a maximum support over the itemsets in S' (i.e., over those that are not explicitly provided in input). Therefore, we provide some extra flexibility to the search algorithm, which can built arbitrary transactions; however, by keeping the support small, we can avoid making these “extra” itemsets frequent in the resulting database. Unfortunately, the introduction of this maximum generic support introduce a further complexity explosion of the problem. In fact, the original decision IFM problem as in [Mie03] as well as several variants introduced by [Cal04, Cal08] are in PSPACE whereas our version with a unique maximum support for all itemsets in S' turns out to be NEXP-complete [].

The negative complexity results did not discourage us for the attempt of finding anyway a solution. Nowadays there is a strong attitude to surrender to intractability or to look for tractable cases even at the cost of futility. Our belief is that intractability must be dealt with more every day and a negative complexity result for a problem cannot refrain us from attempting to solve it. Actually most of the data mining problems are indeed intractable but yet there are a lot of approaches aimed at finding either solutions for small-sized instances or approximated solution. Passing to a more general context, since centuries human brain effectively solves NP-hard problems and it has been a fortune that complexity theory did not have the time to discourage it to do so. The theory cannot be a curb to search solutions but a powerful tool for driving the search. Having this clear in mind, we then embarked on a long journey thru the following stages:

1. We first singled out a particular parameter k measuring the level of overlapping of the input itemsets in S and define a k -parameterized definition of our problem.
2. We showed that for “small” values of k , an instance of our problem can be re-conducted to an instance of the original IFM problem, where S is extended with some of the itemsets in S' (forming a set we call $B_{S'}$), whose number is polynomial in the initial input and in k . The surprising result is that this number is exponential only for very artificially constructed instances with a very intricate overlapping of the itemsets in S , thus the jump to NEXP does not really happen in practice
3. Once landed back to NEXP, we were still in anguish: even coping with NEXP is a very tough adventure! So we formulated the problem as an integer linear program (ILP) with a polynomial number of constraints (corresponding to the supports of the itemsets in $S \cup B_{S'}$) and an exponential number of variables x_j (corresponding to the occurrence number of every possible itemset j) — the ILP input is obviously given in a succinct representation. As the relaxation of the integer constraint does make sense in this case, we can replace ILP with a linear program (LP) whose decision version turns out to be NP-complete.
4. We have crisscrossed two stormy complexity oceans, but the complexity of NP is not at all a quiet sea. We then have given a careful look to the behavior of the classical old simplex algorithm, being aware it continues to be effectively and efficiently used to solve LP problems notwithstanding it may get exponential time. The reason for the simplex to be still largely used depends from the fact the exponential time only arises for few cases. In a sense, there must be some input parameter responsible for a possible exponential execution but the value of such a parameter is small for most of the classical applications of LP — indeed exponential time has been recognized only for some artificial instances. We than decided to board the ship of the simplex method to eventually find a solution to our problem.

5. The final stage is now applying the simplex to a LP with a polynomial number of constraints and an exponential number of variables. The literature gives an interesting solution

Within the extended framework for inverse frequent itemset mining discussed above, our contribution is then to study an efficient and effective solution approach based on linear programming. In more detail,

- ▶ We first propose a novel formulation of the inverse frequent set mining problem, where the size of the database is fixed beforehand and some itemsets are constrained to be infrequent. Computation of the problem is still NP-hard and, hence, unlikely to be efficiently solvable.
- ▶ We describe how the κ -IFM $_{\sigma'}$ can be encoded as an integer linear program (ILP). The multi-criteria version of the κ -IFM $_{\sigma'}$ is presented as an extension of this ILP.
- ▶ We shown how a relaxed version of the κ -IFM $_{\sigma'}$ can be solved by column generation which is a classical technique of large-scale linear programming. The proposed method requires at each step the solution of an auxiliary integer linear program which we prove being NP-hard. We devise an improved formulation and a constructive heuristic for this auxiliary problem. Moreover, the devised solution method is also able to tackle the multi-criteria version of the κ -IFM $_{\sigma'}$.
- ▶ Finally, we conduct a thorough experimental activity over both syntectic and real-life data. Results evidence that our solution approach emerged to be effective for real application scenarios. In addition, the results show that our approach enjoys very good scaling, thereby paving the way for its application over real-life scenarios.

Organization. The chapter is organized as follows. Section 3.2 defines (i) the formal setting of the inverse frequent itemset mining problem, and (ii) its parametrization with respectively complexity, (iii) describes how it can be formulated as an integer linear programming task of reasonable size, and (iv) shows the integer relaxing approximation. In Section 3.3 we show the column generation technique applied on our problem and the relative pricing problem. In particular, for the pricing problem, we show the complexity analysis and solution methods. The efficiency and effectiveness of our approach is illustrated in Section 3.4 and the related works are discussed in Section 3.5. Finally, a set of relevant application are discussed in Section 3.6.

3.2 Problem Formalization

Let $\mathcal{I} = \{o_1, \dots, o_n\}$ be a finite domain of elements, also called *items*. Any subset $I \subseteq \mathcal{I}$ is called an itemset over \mathcal{I} . The universe of itemsets $\mathbf{U}_{\mathcal{I}}$ is the set of all non-empty itemsets over \mathcal{I} . A database \mathcal{D} over \mathcal{I} is bag of itemsets, each one usually called *transaction*. The number of transactions in \mathcal{D} is denoted by $|\mathcal{D}|$. Given a database \mathcal{D} over \mathcal{I} , for each itemset I over \mathcal{I} ($I \in \mathbf{U}_{\mathcal{I}}$), the *support* of I , denoted by $\sigma^{\mathcal{D}}(I)$, is the number of transactions containing I , and the number of duplicates of I , denoted by $\delta^{\mathcal{D}}(I)$, is the the number of transactions equal to I . We say that I is a *frequent* itemset in \mathcal{D} w.r.t. a given support threshold s if $\sigma^{\mathcal{D}}(I) \geq s$. Observe that supports are also represented in the literature as a percentage w.r.t. the dimension of \mathcal{D} , i.e. by $\sigma^{\mathcal{D}}(I)/|\mathcal{D}|$. Finding all the frequent itemsets in \mathcal{D} is the well-known *frequent itemset mining problem*. The *anti-monotonicity property* holds for supports: given two itemsets I and J with $I \subset J$, $\sigma^{\mathcal{D}}(J) \leq \sigma^{\mathcal{D}}(I)$.

We denote the set of natural numbers by \mathbb{N}_0 that will be used for bound. We also introduce the symbol ∞ to denote an unlimited bound and define $\tilde{\mathbb{N}}_0$ as $\mathbb{N}_0 \cup \{\infty\}$ — we therefore

assume that for each $i \in \mathbb{N}_0$, $i < \infty$ holds. Finally, we denote the set of pairs $\{(a, b) : a \in \mathbb{N}_0, b \in \tilde{\mathbb{N}}_0, a \leq b\}$ by $\tilde{\mathbb{N}}^2$.

Thanks to the above notation, we now formally introduce the κ -IFM $_{\sigma'}$ problem:

Definition 3.1. Let:

- i \mathcal{I} be a given set of n items and $\mathbf{U}_{\mathcal{I}}$ be the set of all non-empty itemsets over \mathcal{I}
- ii S be a given set of m itemsets over the items in \mathcal{I}
- iii S' denotes $\{I \in \mathbf{U}_{\mathcal{I}} \mid \nexists I' \in S : I \subseteq I'\}$
- iv $\Gamma_{\sigma} = \{(I, \sigma_{min}^I, \sigma_{max}^I) \mid I \in S, (\sigma_{min}^I, \sigma_{max}^I) \in \tilde{\mathbb{N}}^2\}$ be a given set of triples assigning a minimum and the maximum support to each itemset in S
- v $\sigma' \in \tilde{\mathbb{N}}_0$ be the maximum support threshold for all itemsets in S'
- vi **size** = $(size_1, size_2) \in \tilde{\mathbb{N}}^2$ be the minimal and the maximal dimension for a database.

Then, the σ' -inverse frequent itemset mining problem on \mathcal{I} , S , Γ_{σ} , σ' and **size**, shortly denoted as IFM $_{\sigma'}$, consists of finding a database \mathcal{D} over \mathcal{I} such that the following conditions hold (or of eventually stating that there is no such a database):

$$\forall I \in S : \sigma_{min}^I \leq \sigma^{\mathcal{D}}(I) \leq \sigma_{max}^I \quad (3.1)$$

$$\forall I \in S' : \sigma^{\mathcal{D}}(I) \leq \sigma' \quad (3.2)$$

$$size_1 \leq |\mathcal{D}| \leq size_2. \quad (3.3)$$

If $\sigma' = \infty$ then the problem is simply called the *inverse frequent itemset mining problem*, shortly IFM. Finally, if $\sigma' = 0$ then the problem is denoted by IFM $_S$. \square

The above constraints do formalize the intuitive points we have discussed in the introduction. Indeed, constraints (3.1) state that σ_{min}^I and σ_{max}^I induce a range of admissible supports for each itemset $I \in S$. Constraints (3.2) impose that each itemset which neither belongs to the set S nor is a subset of some itemset in S must have a support $\sigma^{\mathcal{D}}(I)$ less or equal than the threshold σ' . Finally, the size of the database, i.e., the total number of transactions, is constrained in a range indicated by the constraint (3.3). Observe that the classical IFM problem does not contemplate constraints (3.2) so that the solution may contain unexpected frequent itemsets beside to the ones in S .

The complexity of the decision versions (i.e., deciding the existence of a solution) of the two problems in Definition 3.1 has been analyzed in the literature and is summarized next.

Fact 1

- i *The decision version of IFM $_{\sigma'}$ is NEXP-complete [].*
- ii *The decision version of IFM is NP-hard and in PSPACE [Mie03, Cal04].*
- iii *The decision version of IFM $_S$ is NP-complete [GSS09a].* \square

Next we shall concentrate on a special case of IFM $_{\sigma'}$ that turns to be computationally equivalent to IFM. To this end, consider the set of the bottom elements of S' : $B_{S'} = \{I \in S' \mid \nexists I' \in S' : I' \subset I\}$, say with cardinality q' . By the anti-monotonicity property, constraints (3.2) can be replaced by:

$$\forall I \in B_{S'} : \sigma^{\mathcal{D}}(I) \leq \sigma' \quad (3.4)$$

Then it is easy to see that, if the number q' of elements in $B_{S'}$ were polynomial in the size n of \mathcal{I} and the size m of S , IFM $_{\sigma'}$ would reduce to IFM by extending S with the elements of $B_{S'}$. But, as shown in [], in general q' is exponential in n and m . Nevertheless, we next present a condition for the problem instances under which q' is instead polynomial in the number of the items that are included in the itemsets in $B_{S'}$. We need some preliminary definitions to describe such a condition:

- $\hat{S} \subseteq S$ is the set of *maximal frequent itemsets*, i.e., $\hat{S} = \{I \in S \mid \nexists I' \in S : I' \subset I\}$ – let $\hat{m} \leq m$ denote the cardinality of \hat{S}
- $\hat{n} \leq n$ is the number of items in $\mathcal{I}_S = \cup_{I \in S} I = \cup_{I \in \hat{S}} I$
- $\forall I \in \hat{S}$ and $\forall i, 2 \leq i \leq \hat{m} - 2$:
 - $\forall J \in \hat{S}, J \neq I$:
 - $\mathcal{I}_{I,J,i} = \{o \mid o \in I \cap J \text{ and } o \text{ also belongs to at least other } i - 2 \text{ itemsets in } \hat{S}\}$
 - note that $\mathcal{I}_{I,J,2} = I \cap J$
 - $n_{I,J,i}$ is equal to $|\mathcal{I}_{I,J,i}|$ if $|\mathcal{I}_{I,J,i}| \geq i$ or to 0 otherwise – obviously $\hat{n} \geq n_{I,J,2} \geq n_{I,J,3} \cdots \geq n_{I,J,\hat{m}-2}$
 - $\mathcal{I}_{I,i} = \cup_{J: n_{I,J,i} \geq i} \mathcal{I}_{I,J,i}$ and $n_{I,i} = |\mathcal{I}_{I,i}|$
- $\forall i, 2 \leq i \leq \hat{m} - 2, n_i = |\cup_{I \in \hat{S}} \mathcal{I}_{I,i}|$ – obviously $\hat{n} \geq n_2 \geq n_3 \cdots \geq n_{\hat{m}-2}$

Finally we define:

$$\tilde{q}'_{\geq 2} = \sum_{i=2}^{\hat{m}-1} \sum_{I,K \in \hat{S}: n_{I,K,i} \geq i} \binom{n_{I,K,i}}{i} (n_{I,i} - n_{I,K,i}) (n_{K,i} - n_{I,K,i}).$$

The following lemma is proved in the appendix.

Lemma 1 *Given an instance inst of κ -IFM $_{\sigma'}$ with n items, m itemsets in S and $0 < \sigma' < \infty$:*

$$q' = |B_{S'}| \leq n - \hat{n} + \frac{\hat{n}(\hat{n}-1)}{2} + \tilde{q}'_{\geq 2}.$$

where \hat{n} and $\tilde{q}'_{\geq 2}$ are as defined above. □

As explained in the Appendix, the itemsets in $B_{S'}$ with cardinality 1 are exactly $n - \hat{n}$, those with cardinality 2 are at most $\hat{n}(\hat{n}-1)/2$, while $\tilde{q}'_{\geq 2}$ is an upper bound for the itemsets with cardinality ≥ 3 .

Given an instance *inst* of IFM $_{\sigma'}$, we define an integer parameter κ that provides an upper bound for $q' = |B_{S'}|$:

$$\kappa(\text{inst}) = \begin{cases} \lceil 100 \cdot \log_{\hat{n}}(\frac{\hat{n}(\hat{n}-1)}{2} + \tilde{q}'_{\geq 2}) \rceil & \text{if } 0 < \sigma' < \infty \\ 0 & \text{if } \sigma' = 0 \vee \sigma' = \infty. \end{cases}$$

As discussed below, a value for $\kappa(\text{inst})$ less than or equal to 300 indicates that the itemsets in \hat{S} do not have a high number of intricate overlaps for $i \geq 2$. So we can say that a value below 300 is "small". The next result shows that q' is polynomially bounded by n and κ .

Proposition 1 *Given an instance inst of κ -IFM $_{\sigma'}$ with n items, m itemsets in S and $0 < \sigma' < \infty$:*

$$q' = |B_{S'}| \leq n - \hat{n} + \hat{n}^{k/100}$$

where $k = \kappa(\text{inst})$, n and \hat{n} are as defined above.

Proof. By using the Lemma 1 we have that:

$$q' \leq n - \hat{n} + \frac{\hat{n}(\hat{n}-1)}{2} + \tilde{q}'_{\geq 2}.$$

To prove the theorem it is sufficient to prove that $\frac{\hat{n}(\hat{n}-1)}{2} + \tilde{q}'_{\geq 2} \leq \hat{n}^{k/100}$. If we substitute k with $100 \cdot \log_{\hat{n}}(\frac{\hat{n}(\hat{n}-1)}{2} + \tilde{q}'_{\geq 2})$ we have that:

$$\frac{\hat{n}(\hat{n}-1)}{2} + \hat{q}'_{\geq 2} = \hat{n}^{\log_{\hat{n}}(\frac{\hat{n}(\hat{n}-1)}{2} + \hat{q}'_{\geq 2})}$$

Since that $\kappa(inst)$ is defined by the ceiling operator $\lceil 100 \cdot \log_{\hat{n}}(\frac{\hat{n}(\hat{n}-1)}{2} + \hat{q}'_{\geq 2}) \rceil$ then

$$\frac{\hat{n}(\hat{n}-1)}{2} + \hat{q}'_{\geq 2} = \hat{n}^{\lceil \log_{\hat{n}}(\frac{\hat{n}(\hat{n}-1)}{2} + \hat{q}'_{\geq 2}) \rceil}$$

It turns out that if $k \geq 300$ then $q' - (n - \hat{n})$ is at most cubic in \hat{n} . In our experiments, the parameter k is at most 221 and then $q' - (n - \hat{n}) \leq \hat{n}^{2.21}$. Actually in almost all cases k is less than 200 so that $q' - (n - \hat{n})$ is less than quadratic in \hat{n} . Probably only very artificial instances get a high value for $\kappa(inst)$.

Next we introduce a parametrized version of $\text{IFM}_{\sigma'}$ using the definition of κ . *Parameterized complexity* is a measure of complexity of problems with multiple input parameters: typically, in addition to the actual input describing a problem instance, a parameter is used a second input to measure the problem complexity in terms of the value assumed by an instance for that parameter. More formally, a *parameterized problem* is a language $L \subseteq \Sigma \times \mathbb{N}_0$, where Σ is the finite alphabet used for representing the problem (see [] for an overview on parameterized problems). The second component is called the parameter of the problem. A parameterized problem L is *fixed-parameter tractable* if the question $(x, k) \in L?$ can be decided in running time $f(k) \cdot |x|^{\mathcal{O}(1)}$, where f is an arbitrary function depending only on k . The corresponding complexity class is called FPT.

Definition 3.2. The language $\{(inst, \kappa(inst)) \mid inst \text{ is an instance of } \text{IFM}_{\sigma'}\}$ is called the κ -bounded σ' -inverse frequent itemset mining problem, $\kappa\text{-IFM}_{\sigma'}$ for short. \square

Proposition 2 *The decision version of $\kappa\text{-IFM}_{\sigma'}$ is not in FPT unless $\mathcal{P} = \text{NP}$ and an instance $(x, \kappa(x))$ of it can be solved in space polynomial in $|x|$ and $\kappa(x)$.*

Proof. We proceed by contradiction to prove that the decision version of $\kappa\text{-IFM}_{\sigma'}$ is not in FPT unless $\mathcal{P} = \text{NP}$. Suppose that $\kappa\text{-IFM}_{\sigma'} \in \text{FPT}$. Then take any instance x of $\text{IFM}_{\sigma'}$. We can reformulate x as an instance $(x, 0)$ of $\kappa\text{-IFM}_{\sigma'}$. Then, as $\kappa(x) = 0$, a solution can be found in polynomial time - contradiction with Fact 1-(iii).

To see that any instance $(x, \kappa(x))$ of $\kappa\text{-IFM}_{\sigma'}$ can be solved in space polynomial in $|x|$ and $\kappa(x)$, observe that $(x, \kappa(x))$ can be seen as an instance of IFM by extending S with the itemsets of $B_{S'}$. Then the proof is a direct consequence of Fact 1-(iii), since the number of additional itemsets is polynomial in the size of x and in $\kappa(x)$ by Proposition 1,

We point out that parameterized complexity has been introduced in the literature to investigate under which conditions an intractable problem can be made tractable. In our case, parameterization is used to make less intractable the $\kappa\text{-IFM}_{\sigma'}$ problem passing from NEXP to PSPACE. In the next section we shall continue our work of scaling down the complexity: indeed our target is NP. At that point we shall eventually concentrate in finding effective heuristics for solving the $\kappa\text{-IFM}_{\sigma'}$ problem.

3.2.1 The $\kappa\text{-IFM}_{\sigma'}$ as an Integer Linear Program

The $\kappa\text{-IFM}_{\sigma'}$ searches for a mapping between each itemset belonging to the universe $\mathbf{U}_{\mathcal{I}}$ and the integers in $\{0, \dots, size_2\}$, which expresses the number of transactions of each itemset occurring in the resulting database \mathcal{D} (complying with the constraints in Definition 3.1). We next encode this problem as an integer linear program.

W.l.o.g., we select any ordering of $\mathbf{U}_{\mathcal{T}}$, say $\{I_1, \dots, I_{2^n-1}\}$ — for example we fix an ordering of items and take a lexicographic order based on it. Then the vector $T = \{1, \dots, 2^n - 1\}$ provides a suitable representation of all possible itemsets. Let $Q = \{i_1, \dots, i_m\}$ and $Q' = \{j_1, \dots, j_{q'}\}$ represent the indices of the itemsets, respectively in S and in $B_{S'}$, listed in the order fixed for $\mathbf{U}_{\mathcal{T}}$. We use a function \mathcal{Q} from Q to $\{1, \dots, m\}$ to return the position of an index in Q , i.e., for each j_k in Q , $\mathcal{Q}(j_k) = k$.

Let l and u be two vectors of m integers such that for each $j \in Q$, $l_{\mathcal{Q}(j)} = \sigma_{min}^{I_j}$ and $u_{\mathcal{Q}(j)} = \sigma_{max}^{I_j}$.

Let x be a vector of $2^n - 1$ non-negative integer variables whose intended meaning is that its i -th coordinate, x_i , denotes the number of duplicates for the transaction I_i . Moreover, consider a $(2^n - 1) \cdot (2^n - 1)$ matrix A where each entry $a_{ij} \in \{0, 1\}$ is associated with the pair of itemsets I_i and I_j with the intended meaning that $a_{ij} = 1$ if and only if $I_j \supseteq I_i$ holds.

We finally introduce a vector v of $2 \times m + 1$ non-negative rational number artificial variables, whose values represent the costs of violating some support constraints. In particular, v_1, \dots, v_m and $v_{m+1}, \dots, v_{2 \cdot m}$ are the costs of violating respectively lower-bound and upper-bound support constraints on the itemsets in S and $v_{2 \cdot m+1}$ is the cost of violating the lower bound constraint on the database size.

We are now ready to recast the problem in terms of the following integer linear program, whose objective function measures the cost of violating the constraints corresponding to the artificial variables:

$$\text{ILP: minimize } \sum_{i=1}^{2m+1} v_i \quad (3.5)$$

subject to

$$v_{\mathcal{Q}(i)} + \sum_{j \in T} a_{ij} x_j \geq l_{\mathcal{Q}(i)} \quad i \in Q \quad (3.6)$$

$$v_{m+\mathcal{Q}(i)} - \sum_{j \in T} a_{ij} x_j \geq -u_{\mathcal{Q}(i)} \quad i \in Q \quad (3.7)$$

$$-\sum_{j \in T} a_{ij} x_j \geq -\sigma' \quad i \in Q' \quad (3.8)$$

$$v_{2 \cdot m+1} + \sum_{j \in T} x_j \geq \text{size}_1 \quad (3.9)$$

$$-\sum_{j \in T} x_j \geq -\text{size}_2 \quad (3.10)$$

$$v_i \in \mathbb{Q}_0 \quad 1 \leq i \leq 2m + 1 \quad (3.11)$$

$$x_i \in \mathbb{N}_0 \quad i \in T. \quad (3.12)$$

Whenever the bound of some of the constraints (3.7), (3.8) and (3.10) is equal to ∞ or the bound of some of the constraints (3.6) and (3.9) is equal to 0, the corresponding constraint is just dropped. The non-negative rational number variables in V are *artificial* in the sense that their role is to absorb possible violations of the constraints (3.9), (3.6) and (3.7): the minimization of its value entails the search for a solution with the minimal number of violations. For example, assume that $\sum_{j \in T} a_{ij} x_j < l_{\mathcal{Q}(i)}$ for a specific $i \in Q$, then the variable $v_{\mathcal{Q}(i)}$ will be forced by (3.6) to a value at least equal to $l_{\mathcal{Q}(i)} - \sum_{j \in T} a_{ij} x_j$, i.e. a value greater than zero. The objective function (3.5) minimizes the values of the artificial variables. Therefore,

the optimal solution of the presented ILP consists in a database (as induced by variables x in the optimal solution) with minimal violation of the minimal database size constraint. We only insert the artificial variables in the Constraints (3.6,3.7,3.9), because we want that the remaining constraints are directly satisfied. This is always possible as an initial feasible solution can be constructed as follows: $v_{2 \cdot m + 1} = size_1$, $v_{Q(i)} = l_{Q(i)}$, $v_{2 \cdot m + Q(i)} = 0$ and $x_j = 0$, $\forall i \in Q, \forall j \in T$.

Let:

- X_{ILP}^{OPT} be a (feasible) solution of ILP and $b[X_{ILP}^{OPT}]$ be the number of non-zero elements in it — thanks to the artificial variables, a solution always exists;
- $X_{ILP}^{OPT_0}$ be a solution X_{ILP}^{OPT} for which the objective function is equal to 0 - instead, the existence of such an optimal solution is not guaranteed;
- $D[X_{ILP}^{OPT_0}]$ be the database defined by a solution for the objective function is equal to 0 (if any).

Next we prove that the ILP defined by the objective function (3.5), and by the constraints (3.6) – (3.12) is instrumental in solving the κ -IFM $_{\sigma'}$.

Proposition 3 *A database D satisfies the κ -IFM $_{\sigma'}$ problem if and only if there exists an optimal solution $X_{ILP}^{OPT_0}$ for which $D = D[X_{ILP}^{OPT_0}]$.*

Proof. To prove the theorem we show how it is possible to build an admissible solution $X_{ILP}^{OPT_0}$ by using a database D solution of κ -IFM $_{\sigma'}$ and vice versa.

The first transformation consists to take each different transaction in D and compute its number of duplicates, since in the ILP model to each different transaction (or itemset) I_j corresponds a variable x_j then such x_j will have value equal to number of duplicates of I_j . Obviously, since the database D exists, then each variables v_i with $1 \leq i \leq 2m + 1$ has value zero.

The inverse transformation consist into take each variable x_j of ILP problem with value greater than zero and insert in the database D a number of transactions I_j equal to the value of x_j . It is important notice that if the optimal solution of ILP problem is not zero then does not exist a database D that satisfy the κ -IFM $_{\sigma'}$ instance, in fact will exist at least one itemset such that its support does not satisfy its constraint, or the size of database does not satisfy its lower bound.

Note that the total number of constraints is $r = 2 \cdot m + q' + 2$ and the total number of variables is $c = 2^n + 2 \cdot m$, i.e., the number of columns (variables) is exponential in the number of rows (constraints). But the problem can be easily represented in a succinct format with size $n \cdot (m + q') + (2 \cdot m + 3) \cdot w$: the indices of the $m + q'$ itemsets in $Q \cup Q'$ can be represented by n bits (one for each item in \mathcal{I}) and each of the $2 \cdot m + 3$ bounds (2 support bounds for each itemset in S , the unique support bound σ' for each itemset in $B_{S'}$ and the 2 size bounds for the database) can be represented by at most w bits. For simplicity we assume that w is a constant so that the size of the succinct representation is simply $n \cdot (m + q') + 2 \cdot m + 3$.

The next complexity results assume that the input includes q' and is given in a succinct format so that the number r of rows is polynomial in the input whereas the number c of columns is exponential. Thus we can easily transform an instance of κ -IFM $_{\sigma'}$ in an instance of IFM by considering the constraints of itemsets in $B_{S'}$ as constraints in S ; and therefore by the Fact 1 we have the following corollary.

Corollary 1

- i* Deciding whether there exists a solution $X_{ILP}^{OPT_0}$ is in PSPACE and it is in NP if and only decision IFM is in NP;

ii if decision IFM \notin NP then there exist instances of ILP for which every $b[X_{\text{ILP}}^{\text{OPT}_0}]$ is exponential in the size of the succinct input representation.

Because of part (ii) of the corollary, we expect that in general $b[X_{\text{ILP}}^{\text{OPT}_0}]$ is exponential. As shown in the next subsection, an approximate solution with a polynomial number of non-zero elements can be obtained by the continuous relaxation of the integer restrictions, i.e. all the variables are allowed to be rational numbers. So an advantage of such an approximation is the capability of returning an approximated solution with polynomially bounded size.

3.2.2 The κ -IFM $_{\sigma'}$ as a Linear Program

The relaxation of the κ -IFM $_{\sigma'}$ can be stated as a linear program with an exponential number of variables but only a polynomial number of constraints:

$$\text{LP: minimize } \sum_{i=1}^{2m+1} v_i \quad (3.13)$$

subject to the constraints (3.6) – (3.11) of the ILP problem and by replacing the constraint (3.12) with;

$$x_i \in \mathbb{Q}_0 \quad i \in T. \quad (3.14)$$

We assume that the input of the above linear program is given in a succinct format and q' is part of the input as for the ILP problem. Let $X_{\text{LP}}^{\text{OPT}}$ be a (feasible) solution of LP (if any) and $X_{\text{LP}}^{\text{OPT}_0}$ be a solution for which the objective function is equal to 0.

Proposition 4

- i Given a solution $X_{\text{LP}}^{\text{OPT}}$ of LP, $b[X_{\text{LP}}^{\text{OPT}}] \leq r = 2 \cdot m + q' + 2$, i.e., the number of non-zero elements in the solution is less than or equal to r .
- ii Given $z \in \mathbb{Q}_0$, deciding whether there exists a solution $X_{\text{LP}}^{\text{OPT}}$ for which the objective function is less than or equal to z is in NP.

Proof.

- i The first proposition directly derive of the fact that a base solution in a linear programming has the same size of the number of constraints.
- ii In order to decide if exist a solution $X_{\text{LP}}^{\text{OPT}}$ for which the objective function is less than or equal to z , it is sufficient only consider the base solution, then in polynomial time with a nondeterministic Turing machine we can guess a base solution i.e. we can guess r different transactions with respectively number of duplicates. Thus, after the guessing of the base, it is sufficient a polynomial time to verify the feasibility of the base and verify that the objective function is less than or equal to z . Therefore the problem is in NP.

Let $D[X_{\text{LP}}^{\text{OPT}}]$ be the database defined by $\text{round}[X_{\text{LP}}^{\text{OPT}}]$, that is obtained from $X_{\text{LP}}^{\text{OPT}}$ by rounding the values of its elements. We point out that ordinary rounding (i.e. the nearest integer) is not necessarily considered for some variables floor or ceiling can be enforced; then it turns out that there are many instances of $\text{round}[X_{\text{LP}}^{\text{OPT}}]$.

Proposition 5 Given an optimal solution $X_{\text{LP}}^{\text{OPT}_0}$ of LP, there exists a rounding for $X_{\text{LP}}^{\text{OPT}_0}$ such that $\text{round}[X_{\text{LP}}^{\text{OPT}_0}]$ is a solution of ILP whose objective function value is less than or equal to $r \cdot (m + 1)$.

Proof. To prove the theorem we use the more simple rounding that is the floor operator $\lfloor \cdot \rfloor$, but is important to notice that there exist more powerful rounding that in practical case allows a more restrictive gap between the objective function with or without rounding. If we consider the round $\lfloor X_{LP}^{OPT_0} \rfloor$ then we have that the $v_{m+Q(i)}$ variables for $i \in Q$, of Constraints 3.7, are equal to zero. Instead the $v_{Q(i)}$ variables for $i \in Q$, of Constraints 3.6, have values at most r , since in each of them constraints are present at most r value great than zero (because of it is a base solution), and the round operator is the floor that produce for each variable an error at most 1. The same thing happens for the variable $v_{2 \cdot m + 1}$ of Constraints 3.9. Thus the gap is at most $r \cdot (m + 1)$ where $(m + 1)$ represent the number of constraints of type 3.6 and 3.9.

Experimental results prove that in practice the approximate solution is instead far below the above bound.

The relaxation of integer constraints opens an interesting perspective for an effective solution of $\kappa\text{-IFM}_{S'}$. But finding an optimal solution of LP is not an easy task because of the exponential number of variables. In the next section we present an algorithm which at each step explicit a polynomial number of variables.

3.3 Column Generation Algorithm to solve LP

Column generation, see e.g [DT03] and [DDS05], is a method of dealing with linear programs with a large number of variables. This method solves a linear program without explicitly including all columns (i.e., variables), in the coefficient matrix but only a subset of them with cardinality equal to the number of rows (i.e., constraints). Columns are dynamically generated by solving an auxiliary optimization problem called the *pricing problem*.

We present and discuss the specialization of the column generation algorithm for our problem in Section 3.3.1. In Section 3.3.2 we prove that the decision version of the pricing problem is NP complete, so it is unlikely that a polynomial algorithm exists for this problem and show that a solution can be computed using an integer linear program. As the PRICE problem is not tractable, we present a heuristic algorithm for the pricing problem in Section 3.3.3. Finally, in Section 3.3.4 we present the heuristic version of the column generation algorithm that will be used in our experiments.

3.3.1 Column Generation Algorithm

The linear program to be solved is denoted as the *master problem* (MP). In our case the MP problem consists of $r = 2 \cdot m + q' + 2$ rows (i.e., the $2 \cdot m$ support constraints for the itemsets in S , the q' support constraints for the itemsets in $B_{S'}$ and the 2 size constraints for the database) and $c = 2^n + 2 \cdot m$ columns (i.e., the $2^n - 1$ variables in T and the $2 \cdot m + 1$ artificial variables). As discussed in Section 3.2.1, the MP problem is represented in a succinct format with size $n \cdot (m + q') + 2 \cdot m + 3$.

The linear program with only a subset of the MP columns with cardinality c' equal to the number r of rows is called the *restricted master problem* (RMP) – it turns out that an RMP problem does not need a succinct representation as r is polynomial in the succinct size of the input. Actually the number of columns c' passed to RMP can be greater than r , provided that c' is polynomial in r . From linear programming theory we know that if there is an optimal solution then there also exists an optimal solution corresponding to a basis of the coefficient matrix (in our case any basis consists of r columns). The columns of an optimal basis are

those strictly necessary, all other columns can be ignored. Thus, to solve the MP is equivalent to solve the RMP with only the columns defining an optimal basis.

The column generation method looks for an optimal basis as within the simplex algorithm. The simplex algorithm starts from an initial basis and moves from a current basis to a new basis by removing from the basis one column and adding to the basis a column with a negative (here and in the following we refer to the minimization case) reduced cost (*iteration step*). Primal feasibility is maintained and the objective function is non-increasing during this search. The reduced cost of a column can be computed by using the current dual variables. If there is not a column with a negative reduced cost, then the simplex algorithm terminates and the current basis is proved to be optimal. This scheme is maintained within the column generation method. However, the task of providing a column with a negative reduced cost, or certifying that there is not such a column, is delegated to the pricing problem.

The pseudo-code of the column generation algorithm is presented in Figure 3.1. The algorithm starts by initializing B , that is the list of variables to be given as input to the method RMP at the first call. We include in B the columns corresponding to all $2 \cdot m + 1$ artificial variables, to the m itemsets in S and to the q' itemsets in $B_{S'}$, i.e., $c' = |B|$ is $r = 3 \cdot m + q' + 1$. As discussed in Section 3.2.1, a feasible solution can be easily found using such columns. The output of RMP is: B' (the list of variables in the computed basis), Z (the list of values for the basis variables), D (the values of the r dual variables). Observe that, in addition to B , the method RMP uses additional data that are necessary to construct the projection of the linear program to the basis variables - such data are taken from the succinct representation of the problem instance.

The method PRICE receives as input D and returns (j, \tilde{c}_j) , where j is a column with a minimum reduced cost \tilde{c}_j . Also this method uses data from the succinct representation of the problem instance to find a column with a negative reduced cost. If \tilde{c}_j happens not to be negative, the current basis is optimal and the "while" cycle stops; otherwise, the column j and all the columns in V are added to the previous basis B' to update B and the cycle continues. Note that adding the columns in V would not be necessary. However, as the number $2 \cdot m + 1$ of artificial variables is polynomial, we also include all such variables in B , so that at each step the column j returned by PRICE corresponds to a variable in Q , whose size is instead exponential — this technicality will help in simplifying notation in the next subsection where the complexity of PRICE will be investigated. After having updated B , a new step of the algorithm is performed and the iterations continue until there exists a column with a negative reduced cost.

The overall algorithm is a particular implementation of the a simplex method and, therefore, it eventually terminates. In the worst-case it has to explore all the bases, whose number is exponential in the number of constraints and variables. In our problem the number of variables is itself exponential, leading to worrying worst-case conditions. As we shall show in Section 4, the number of iterations is such that the algorithm can have practical use. This is coherent with the practice were the simplex algorithm shows a remarkably fast convergence. The real problem is the implementation of the method PRICE. As we shall prove in the next sub-section, the function PRICE cannot be computed in polynomial time unless $\mathcal{P} = \text{NP}$.

Despite the intractability of the function PRICE, the column generation algorithm has an attractive characteristic: columns that exit the basis can be dropped from the RMP. By so doing the algorithm has bounded use of the space, proportional to the number of constraints, i.e. the dimension of a basis. Therefore, by using a heuristic polynomial-time implementation of PRICE, the column-generation algorithm may act as the simplex method for which each iteration is executed in polynomial time although the number of variables is exponential.

Input: Succinct representation of LP.
Output: B' (list of variables in the solution basis), Z (list of values for the basis variables)
Algorithm:

```

1 Initialize  $B = V \cup Q \cup Q'$  and set STOP = false;
2 while not STOP do
3    $(B', Z, D) := \text{RMP}(B)$ ;
4    $(j, \tilde{c}_j) := \text{PRICE}(D)$ ;
5   if  $\tilde{c}_j < 0$  then
6      $B = B' \cup V \cup \{j\}$ ;
7   else
8     set STOP = true;
9   end if
10 end while
11 return  $(B', Z)$ 

```

Fig. 3.1. Column Generation Algorithm

3.3.2 Complexity of the Pricing Problem

We are given a set D of dual variable rational number values. We represent them by the m -element vectors λ and π , the q' -element vector ξ , and the scalars τ_1 , and τ_2 of the RMP, that are associated to the constraints (3.6), (3.7), (3.8), (3.9) and (3.10) respectively.

Given a column j corresponding to any itemset variable in T , the reduced cost \tilde{c}_j is:

$$\begin{aligned} \tilde{c}_j &= 0 - (\tau_1 - \tau_2 + \sum_{i \in Q} a_{ij} \lambda_i - \sum_{i \in Q} a_{ij} \pi_i - \sum_{i \in Q'} a_{ij} \xi_{i-m}) \\ &= -\tau_1 + \tau_2 + \sum_{i \in Q} a_{ij} (\pi_i - \lambda_i) + \sum_{i \in Q'} a_{ij} \xi_{i-m}. \end{aligned} \quad (3.15)$$

For notational simplicity, we define $R = Q \cup Q'$, $\tau = \tau_1 - \tau_2$ and ϕ as the $(m + q')$ -element vector $\begin{pmatrix} \pi - \lambda \\ \xi \end{pmatrix}$. Then, as $a_{ij} = 1$ if $I_i \subseteq I_j$ or $a_{ij} = 0$ otherwise, where the itemsets I_i and I_j correspond respectively to the columns i and j , the reduced cost can be reformulated as:

$$\tilde{c}_j = -\tau + \sum_{i \in R} a_{ij} \phi_i = -\tau + \sum_{i \in R, I_i \subseteq I_j} \phi_i.$$

We are now ready to define the *PRICE problem* and its decisional version.

Definition 3.3 (PRICE Problem).

Given R , ϕ and τ , the *PRICE problem* consists in finding a column j in T such that $\tilde{c}_j \leq \tilde{c}_i$, $\forall i \in T$.

The *Decisional PRICE problem* is: given a rational number α , does there exist a column j in T such that $\tilde{c}_j \leq \alpha$. \square

Theorem 3.4. *The Decisional PRICE problem is NP-complete*

Proof. The Decisional Price is in NP because a non-deterministic Turing machine can guess in polynomial time a I_j contained in \mathcal{I} and verify in polynomial time that $\tilde{c}_j < 0$.

To prove the hardness of the problem, we next exhibit a reduction from the *graph 3-colorability problem*. This problem consists in deciding whether, given a graph $G = (N, E)$, there is a 3-coloring $col : N \rightarrow \{r, g, b\}$ such that $col(i) \neq col(j)$ for each adjacent pair of nodes, i.e. $(i, j) \in E$.

Based on the input graph $G = (N, E)$, we construct an instance of the *Decisional Price* problem such that: the set \mathcal{I} of items is $\{r_w | w \in N\} \cup \{g_w | w \in N\} \cup \{b_w | w \in N\}$, where conceptually the items r_w, g_w, b_w are the colors that can be associated to each node in N , $\tau = -(|N| - 1)$ and R contains all index of the following two groups of itemsets. For each node $w \in N$, there are the following itemsets—**Group (I)**:

- $I_{i_w,1} = \{r_w\}$ and $\phi_{i_w,1} = -1$
- $I_{i_w,2} = \{b_w\}$ and $\phi_{i_w,2} = -1$
- $I_{i_w,3} = \{g_w\}$ and $\phi_{i_w,3} = -1$
- $I_{i_w,4} = \{r_w, b_w\}$ and $\phi_{i_w,4} = 3 \cdot |N|$
- $I_{i_w,5} = \{b_w, g_w\}$ and $\phi_{i_w,5} = 3 \cdot |N|$
- $I_{i_w,6} = \{r_w, g_w\}$ and $\phi_{i_w,6} = 3 \cdot |N|$

Moreover, for each edge $(w, v) \in E$, there are the following itemsets—**Group (II)**:

- $I_{i_w,v,7} = \{r_w, r_v\}$ and $\phi_{i_w,v,7} = 3 \cdot |N|$
- $I_{i_w,v,8} = \{b_w, b_v\}$ and $\phi_{i_w,v,8} = 3 \cdot |N|$
- $I_{i_w,v,9} = \{g_w, g_v\}$ and $\phi_{i_w,v,9} = 3 \cdot |N|$

Note that the itemsets in **Group (I)** model each of the three possible assignments of a color to any node in the graph. Instead, itemsets in **Group (II)** guarantee that two adjacent nodes in G cannot be colored with the same color. It is easy to note that for each coloring col_h of G is associated a unique column h and if col_h is a valid coloring for G then $\tilde{c}_h = -1$ otherwise $\tilde{c}_h \geq 0$. Hence, the Decisional Price is NP-complete in general.

Let \tilde{T} be the set of columns j corresponding to itemsets obtained by the union of a number of itemsets in R , thus $\tilde{T} = \{j \mid \exists j_1, \dots, j_p \in R \text{ s.t. } I_j = I_{j_1} \cup \dots \cup I_{j_p}\}$. The next result shows that the search for the column j with minimum reduced cost \tilde{c}_j can be restricted to the columns in \tilde{T} .

Proposition 6 *Let \tilde{c} be the minimum reduced cost. Then there exists a column j in \tilde{T} such that $\tilde{c}_j = \tilde{c}$*

Proof. Let $I_y \in U_{\mathcal{I}}$ be a solution with minimum reduced cost \tilde{c} , we can build an itemset I_j in the following way:

$$I_j = \bigcup_{i \in R, I_i \subseteq I_y} I_i.$$

By definition of reduced cost we have that $\tilde{c}_j = \tilde{c}$. Since the construction of I_j is a union of some itemsets with index in R the I_j belongs to \tilde{T} .

In order to give an effective procedure for solving the PRICE problem, next we formulate it in terms of an integer linear program that computes an itemset U^* , say with index j , such that $j \in \tilde{T}$ and \tilde{c}_j is minimum. U^* is represented by a vector of binary variables $\beta = [\beta_1, \dots, \beta_n]^T$, corresponding to the n items: each component β_h indicates whether U^* contains the item o_h ($\beta_h = 1$) or not ($\beta_h = 0$). We use the vector of binary variables $y = [y_1, \dots, y_{|R|}]^T$, corresponding to the itemsets in R , to model the inclusion of such itemsets in U^* : thus, if $I_i \subseteq U$ then $y_i = 1$, otherwise $y_i = 0$. Then U^* is the union of all itemsets i in R for which $y_i = 1$.

The ILP formulation to solve the pricing problem, denoted as \mathcal{F}_1 , follows:

$$\text{minimize } \sum_{i \in R} \phi_i y_i \quad (3.16)$$

$$\sum_{o_h \in I_i} \beta_h + 1 \leq |I_i| + y_i \quad i \in R, \quad (3.17)$$

$$\beta_h \leq \sum_{i \in R, o_h \in I_i} y_i \quad h \in \{1, \dots, n\}, \quad (3.18)$$

$$y_i \leq \beta_h \quad i \in R, o_h \in I_i, \quad (3.19)$$

$$y_i \in \{0, 1\} \quad i \in R, \quad (3.20)$$

$$\beta_h \in \{0, 1\} \quad h \in \{1, \dots, n\}. \quad (3.21)$$

The objective function (3.16) represents the reduced cost \tilde{c}_j (modulo the constant τ) of a generic column $j \in \tilde{T}$, that we want to minimize to compute U^* . The constraints (3.17) impose that, given any $i \in R$, if $\beta_h = 1$ for all $o_h \in I_i$ then $y_i = 1$; in other words, if U^* contains all items of an itemset $i \in R$, then I_i must be declared to be an itemset included in U^* . The constraints (3.18) impose that if $\beta_h = 1$ then $\exists i \in R : o_h \in I_i \wedge y_i = 1$; thus, an item o_h is in U^* only if at least one of the itemsets included in U^* contains o_h . The constraints (3.19) impose that if $\exists o_h \in I_i : \beta_h = 0$ then y_i must be equal to zero, i.e., an itemset I_i cannot be declared included in U^* if any of its items is not contained in U^* . The constraints (3.19) also enforce that if I_i is declared to be an itemset included in U^* ($y_i = 1$) then all items of it must be in U^* as well ($\beta_h = 1, \forall h \in I_i$).

Proposition 7

- \mathcal{F}_1 solves the PRICE problem;
- \mathcal{F}_1 solves the PRICE problem also if the constraints (3.20) are replaced by:

$$0 \leq y_i \leq 1 \quad i \in R. \quad (3.22)$$

Proof.

- To prove that \mathcal{F}_1 solves the PRICE problem straightforward from the previous description of \mathcal{F}_1 formulation;
- We prove that it is sufficient imposes the integer constraints only on the variables β . In fact for each $i \in R$ we have that two case are possible: (i) each variable β_h with $o_h \in R$ has value 1, or (ii) exist at least one variable β_h with $o_h \in R$ that has value 0. In the first case, by Constraints 3.17 the value of variable y_i is one. Instead in the second case, by Constraints 3.19 the value of variable y_i is zero. Thus It is not necessary the integer constraints over the y variables because if the β variables are integer variables then variables y can be one or zero.

It turns out that if we relax the binary integer constraints (3.20) the formulation remains correct and if $n \gg m$ this relaxation can increase the performance of the ILP solver.

3.3.3 A Heuristic Algorithm for the Pricing Problem

In the following we present a heuristic algorithm for the pricing problem that runs in polynomial time. The heuristic starts by computing an initial (possibly empty) itemset $U_0 \subseteq \mathcal{I}$, computed as the union of all itemsets $I_i, i \in R$, with $\phi_i < \phi^0 = \max(0, \tau)$, thus:

$$U_0 = \bigcup_{i \in R, \phi_i < \phi^0} I_i \quad (3.23)$$

Observe that, even though U_0 is constructed by the union of all itemsets i in R with $\phi_i < \phi^0$, it may eventually happen that U_0 also contain also some itemset $i \in R$ with $\phi_i \geq \phi^0$.

Proposition 8 *If the the minimum reduced cost \tilde{c} is negative then there exists U^* with reduce cost \tilde{c} such that $U^* \subseteq U_0$*

Proof. By using the Proposition 6 without loss of generality we can consider only the itemset with index in \tilde{T} . Let I_j with $j \in R$ be an itemset with $\tilde{c}_j = \tilde{c}$ and such that does not exist an other index $x \in \tilde{T}$ such that $\tilde{c}_x = \tilde{c}$ and $I_x \subset I_j$. Suppose now that I_j is not contained in U_0 , therefore exist at least one itemset I_y with $y \in R$ contained in I_j that is not contained in U_0 . Obviously, since I_j is not in U_0 then $\tilde{c}_j > 0$. Thus the itemset $I_j \setminus (I_j \setminus U_0)$ has a reduced cost less than \tilde{c} and I_j cannot be optimal.

It turns out that if U_0 is empty then the reduced cost of U^* is not negative.

We restrict the search space for U^* to the itemsets U that are subsets of U_0 . The outline of the heuristic is presented in Figure 3.2. We start by setting $U = U^+ = U_0$, where U^+ is the current sub-optimum (actually, heuristic optimum). At each iteration, in a greedy fashion, we reduce the size of the current itemset U of one item. The search of the best item $o \in U$ to remove from U is done by minimizing the function $cost(U \setminus \{o\})$, returning the reduced cost (modulo the constant $-\tau$) of the column associated to the itemset U :

$$cost(U) = \sum_{i \in R, I_i \subseteq U} \phi_i. \quad (3.24)$$

(The cost is 0 if there are no itemsets i satisfying the conditions of the sum.) Note that, after the removal of an item o , the itemset $U' = U \setminus \{o\}$ may contain some items that are contained in none of the itemsets i in R that are included in U' . Because of Proposition 6, all such items can be removed. So, the algorithm replaces U' by

$$reduce(U') = \bigcup_{i \in R, I_i \subseteq U'} I_i.$$

Obviously $cost(U') = cost(reduce(U'))$; moreover, if $U^* \subseteq U'$ then $U^* \subseteq reduce(U')$ as well.

We then update the current itemset U by setting $U = reduce(U \setminus \{o\})$ (step 6); in addition, if its cost is lower than the one of the current sub-optimum U^+ , U becomes the new sub-optimum (step 8). The *while* cycle stops when either U becomes empty or $cost_{<0}(U) < cost(U^+)$ where $cost_{<0}(U)$ is the sum of ϕ_i values of all itemsets i in R that are contained in U and for which ϕ_i is negative:

$$cost_{<0}(U) = \sum_{i \in R, I_i \subseteq U, \phi_i < \phi^0} \phi_i \quad (3.25)$$

Note that $cost_{<0}(U)$ is always less than or equal to $cost(U)$. Moreover, it represents a lower bound for the costs for subsets of U :

Input: Set H of itemsets over \mathcal{I} , function Φ .
Output: U^+ subset of \mathcal{I} .
Algorithm:

- 1 compute $U_0, C_{U_0} = \text{cost}(U_0)$ and $C_{<\phi^0, U_0} = \text{cost}_{<\phi^0}(U_0)$;
- 2 $U = U_0; C_U = C_{U_0}; C_{<\phi^0, U} = C_{<\phi^0, U_0}$;
- 3 $U^+ = U; C_{U^+} = C_{U_0}$;
- 4 **while** ($U \neq \emptyset$ and $C_{<\phi^0, U} < C_{U^+}$)
- 5 $o^* = \arg \min_{o \in U} (\text{cost}(U \setminus \{o\}))$;
- 6 $U = \text{reduce}(U \setminus \{o^*\}); C_U = \text{cost}(U); C_{<\phi^0, U} = \text{cost}_{<\phi^0}(U)$;
- 7 **if** ($C_U < C_{U^+}$)
- 8 $U^+ = U; C_{U^+} = C_U$;
- 9 **end if**
- 10 **end while**
- 11 **return** U^+ ;

Fig. 3.2. Heuristic PRICE Algorithm

$$\text{cost}_{<\phi^0}(U) \leq \text{cost}(\hat{U}), \forall \hat{U} \subseteq U$$

Obviously, if $\text{cost}_{<\phi^0}(U) \geq \text{cost}(U^+)$ then it is not possible to find a subset of U that has a cost less than or equal to $\text{cost}(U^+)$; therefore, we can stop the iterations and return the current sub-optimum U^+ .

Proposition 9 *The heuristic PRICE algorithm runs in time $\mathcal{O}(n^3 \times (m + q'))$.*

Proof. To compute U_0 (step 1) it is sufficient to verify for each itemset i in R whether $I_i \subseteq U_0$ or not and, if the test is successful, to include the items of I_i in U_0 (*cost procedure*). As the cardinality of R is $m + q'$ and each inclusion test can be easily implemented in time linear in the total number n of items, the computation of U_0 can be done in time $\mathcal{O}(n \times (m + q'))$. With some minor variations to the cost procedure, also computing $\text{cost}(U_0)$ and $\text{cost}_{<\phi^0}(U_0)$ (step 1) as well $\text{reduce}(U \setminus \{o^*\})$ in $\text{cost}(U)$, $\text{cost}_{<\phi^0}(U)$ (step 6) can be done in the same time. Step 5 requires to repeat the cost procedure at most n times and, then, has a time complexity $\mathcal{O}(n^2 \times (m + q'))$. This step is repeated at most n times since at least one item is dropped out at each "while" iteration, So the overall algorithm runs in time $\mathcal{O}(n^3 \times (m + q'))$.

3.3.4 Heuristic Column Generation Algorithm

Let us now turn our attention to the exact Column Generation Algorithm described in Figure 3.1. The method *PRICE* at step 4 computes an exact solution of the pricing problem; so we can think that the method is an implementation of a solver for the ILP problem defined in Section 3.3.2. A straightforward heuristic version of the Column Generation Algorithm could only consist in replacing *PRICE* with the method *HPRICE* implementing the heuristic price algorithm described in Figure 3.2. But, then, if the reduced cost returned by *HPRICE* is not negative, we are not anymore guaranteed that we have found the optimal solution. Therefore, instead of stopping the iteration, we decide to call at this point the exact pricing procedure. As this procedure is computationally expensive, we introduce an alternative stop criterion: if

Input: Succinct representation of LP, a global time limit, an ILP time limit ITL , a unfeasibility tolerance.

Output: B' (list of variables in the solution basis), Z (list of values for the basis variables)

Algorithm:

```

1 Initialize  $B = V \cup Q \cup Q'$  and set  $STOP = false$ ;
2 while not  $STOP$  and the global time limit has not been reached do
3    $(B', Z, D) := RMP(B)$ ;
4    $(j, \tilde{c}_j) := HPRICE(D)$ ;
5   if  $\tilde{c}_j < 0$  then
6      $B = B' \cup V \cup \{j\}$ ;  $FOUND = true$ ;
7   else
8     if the unfeasibility tolerance is satisfied then;
9      $STOP = true$ ;
10  else
11     $(j, \tilde{c}_j) := PRICE(D, ITL)$ ;
12    if  $\tilde{c}_j < 0$  then
13       $B = B' \cup V \cup \{j\}$ ;
14    else
15       $STOP = true$ ;
16    end if
17  end if
18 end if
19 end while
20 return  $(B', Z)$ 

```

Fig. 3.3. Heuristic Column Generation Algorithm

the heuristic procedure fails in finding a new column with negative reduced cost and we have reached a satisfactory, user defined, level of unfeasibility we can end the algorithm. Recall that, due to the artificial variables, a solution different from 0 for the objective function implies that some constraints are not satisfied, i.e., the solution is unfeasible. It does then make sense to fix a satisfactory level of unfeasibility. This is further motivated by the fact that, because of the LP relaxation of the original ILP problem, some constraints would not be satisfied even when the objective function is equal to 0.

In case we are obliged to call the exact pricing procedure, to avoid excessive computational times, we introduce a user-defined time limit ILP for its execution. But a high computational cost could also derive by the fact that the simplex algorithm, in the worst-case, would explore a very large number of the feasible bases. As discussed in Section 3.3.1, in practice the simplex algorithm shows a remarkably fast convergence. However, in terms of worst-case computational complexity, we cannot rule out the possibility of getting excessive computational times. Therefore, we have to provide a global time-limit for termination.

The heuristic column generation algorithm that includes all variation discussed above is presented in Figure 3.3. The algorithm stops for one of the the following three conditions: (i) the global time-limit has been reached, (ii) the heuristic pricing algorithm fails and a satisfactory level of unfeasibility is reached and (iii) the exact pricing algorithm does not return a

column with negative reduced cost. The latter condition indicates that the current solution is optimal whenever the exact pricing algorithm does not terminate because of the imposed time limit.

3.4 Computational results

We now present quantitative results to assess the effectiveness of the proposed solution method. We first describe how we have built test instances, and we then discuss extensive computational experiments.

3.4.1 Test instances

Datasets

We experiment our solution method on problem instances for which known solutions (i.e. the original dataset itself from which they are taken) exist. To this purpose, we refer to three distinct datasets², which are often used as benchmarks for frequent itemsets discovery algorithms [ZKM01]: the artificial dataset T10I4D100K, that is generated by the *IBM Almaden* association rule synthetic data generator, and the real datasets BMS-WebView-1 and BMS-WebView-2, that contain clickstream data from two e-commerce web sites. In the latter two datasets, each itemset represents a web session consisting of all the product detail pages viewed in that session and each product detail view represents an item.

The characteristics of these three datasets are described in Fig. 3.4. The first column of Fig. 3.4 indicates the name of the dataset, and the other columns report (from left to right) the following data: the number of itemsets $|\mathcal{D}|$ (i.e., the size of the dataset), the maximum itemset size, the average itemset size, the total number of items $n = |\mathcal{I}|$, the average frequency of an item in the itemsets, and the number of distinct itemsets (i.e., without duplicates).

\mathcal{D}	$ \mathcal{D} $	Max. Trans. Size	Avg. Trans. Size	$n = \mathcal{I} $	Avg. Item Freq.	Num. Dist. Trans.
BMS-WebView-1	59602	267	2.5	497	0.0050	18473
BMS-WebView-2	77512	161	4.6	3340	0.0013	48684
T10I4D100K	100000	300	10.0	870	0.0116	89135

Fig. 3.4. Characteristics of the three datasets used to generate test instances.

Basic test instances

A number of test instances are built as follows. We call them basic as we shall later derive additional instances from them.

For each dataset \mathcal{D} , we extract—by using standard itemsets discovery algorithms—the set S of all itemsets that are frequent in \mathcal{D} w.r.t. a given threshold s . We express the threshold s in

² Datasets can be downloaded from the KDD-Cup 2000 competition website [KDD].

percentage points w.r.t. the number of itemsets in the dataset. We use values of the threshold s belonging to the set $\{0.2\%, 0.3\%, \dots, 0.9\%, 1\%\}$. These very low support thresholds are necessary to yield a significant number of frequent itemsets, since for $s > 1\%$ there are few frequent itemsets in the original datasets, see [ZKM01, KDD] for a detailed description of the datasets.

In order to characterize the unfrequent itemsets, for each s we need to fix a suitable value for σ' , that is the maximum support that each itemset not belonging to S can have in order to prevent undesired frequent itemsets. Obviously, $100 \times \sigma' / |\mathcal{D}|$ must be less than s . So it would be sufficient to set σ' equal to the greatest integer number that divided for the size of \mathcal{D} is smaller than $s \cdot 10^{-2}$; we call this value σ'_{max} . However, in order to prevent rounding errors, we decrease σ'_{max} of a small factor equal that, on the basis of our experiments, we have fixed to $4 \cdot 10^{-4} |\mathcal{D}|$. In sum $\sigma' = \sigma'_{max} - 4 \cdot 10^{-4} |\mathcal{D}|$.

So we have constructed 9 instances (one for each of the values of s) for each of the 3 datasets: in total 27 instances. For each instance $(\mathcal{I}, S, \Gamma_\sigma, \sigma', size)$, \mathcal{D} must satisfy the constraint Γ_σ , that contains a triple of the form $(I, \sigma'_{min}, \sigma'_{max})$ for each $I \in S$. We set $\sigma'_{min} = \sigma'_{max} = \sigma^{\mathcal{D}}(I)$, where $\sigma^{\mathcal{D}}(I)$ is the actual support of the itemset I in the database \mathcal{D} .

The table of Fig. 3.5 reports the cardinality of the sets S and $B_{S'} = Q'$ in the generated instances according to the original dataset and the threshold s . We recall that the cardinality q' of Q' is the number of constraints (3.8) in Section 3.2.1. On the other hand the number of both groups of constraints (3.6) and (3.7) is equal to the cardinality m of S . The values of m , q' and n characterize our basic test instances of κ -IFM $_{\sigma'}$.

	BMS-WebView-1 $n = 497$		BMS-WebView-2 $n = 3340$		T10I4D100K $n = 870$	
$s(\%)$	m	q'	m	q'	m	q'
0.20	798	36506	3683	167688	13255	273436
0.30	435	25530	1340	61083	4552	238332
0.40	286	16605	676	30343	2001	197796
0.50	201	11498	408	17642	1073	161617
0.60	162	8732	257	11255	772	133080
0.70	133	6260	187	8614	603	113368
0.80	105	4226	138	6927	494	98294
0.90	90	3260	113	6051	421	81858
1.00	77	2633	81	4827	385	70611

Fig. 3.5. Values of $m = |S|$ and $q' = |Q'|$ in the basic test instances.

Boundedness of $B_{S'}$

In Table 3.6 we show the values of \hat{n} (i.e., the total number of items that are included in the itemsets in S), of \hat{m} (i.e., the number of maximal itemsets in S) and of the values of $k = \kappa(inst)$ for each basic test instance $inst$. The highest values for k are taken by the instances corresponding to the lowest support $s = 0, 2\%$.

We point out that all the values of k are indeed small in the sense that the estimated number \tilde{q}' of itemsets in Q' is a bit more than quadratic in \hat{n} for a few instances whereas in general is less than quadratic. Recall that $\tilde{q}' - (n - \hat{n}) = \hat{n}^{k/100}$, where $\tilde{q}' - (n - \hat{n})$ is

	BMS-WebView-1 $n = 497$			BMS-WebView-2 $n = 3340$			T10I4D100K $n = 870$		
s (%)	\hat{n}	\hat{m}	k	\hat{n}	\hat{m}	k	\hat{n}	\hat{m}	k
0.2	268	483	181	573	1257	221	741	1938	200
0.3	225	296	188	340	617	202	692	1293	190
0.4	181	217	187	233	379	193	630	761	190
0.5	150	162	187	170	251	189	569	585	190
0.6	130	136	186	127	173	187	516	511	189
0.7	109	115	186	104	130	186	476	455	189
0.8	88	90	185	86	108	185	443	432	189
0.9	76	76	184	75	91	184	404	397	189
1.0	67	67	184	56	67	183	375	370	189

Fig. 3.6. Values of $\hat{m} = |\hat{S}|$, $\hat{n} = |\cup_{I \in \hat{S}} I|$ and of the parameter k

the estimated number of singleton itemsets in Q' and $\hat{n}^{k/100}$ is the estimated number of those itemsets with at least two items. The latter value is in great part determined by $n \cdot (n - 1)/2$, that is the estimated number of itemsets with exactly two items. This confirms our belief that q' is exponential only for artificial datasets having strange overlaps of the itemsets in \hat{S} .

Table 3.7 compare the values of $k/100$ with the exponent d of the formula $q' - (n - \hat{n}) = \hat{n}^d$, where q' is the actual number of itemsets in Q' . The values of $k/100$ practically coincide with those of d for all instances except the one of the dataset T10I4D100K for $s = 0.2\%$ and the ones of BMS-WebView-2 for $s = 0.2, 0.3, 0.4$. Observe that the number of itemsets in Q' with two items is at most $n \cdot (n - 1)/2$. So it is interesting to compare the values of $k/100$ and d with $e = \log_{\hat{n}}(\hat{n} \cdot (\hat{n} - 1))$. As depicted in the same table, the three values practically coincide in all most cases. Probably $(n - \hat{n}) + \hat{n} \cdot (\hat{n} - 1)$ gives a good estimation of q' in most real problem instances.

	BMS-WebView-1			BMS-WebView-2			T10I4D100K		
s (%)	k	d	e	k	d	e	k	d	e
02	1.89	1.88	1.88	2.21	1.90	1.90	2.00	1.90	1.90
03	1.88	1.88	1.88	2.02	1.89	1.89	1.90	1.90	1.90
04	1.87	1.87	1.87	1.93	1.88	1.88	1.90	1.90	1.90
05	1.87	1.86	1.87	1.89	1.87	1.87	1.90	1.90	1.90
06	1.86	1.86	1.86	1.87	1.86	1.86	1.89	1.89	1.89
07	1.86	1.85	1.86	1.86	1.85	1.85	1.89	1.89	1.89
08	1.85	1.85	1.85	1.85	1.85	1.85	1.89	1.89	1.89
09	1.84	1.84	1.84	1.84	1.84	1.84	1.89	1.89	1.89
1	1.84	1.84	1.84	1.83	1.83	1.83	1.89	1.89	1.89

Fig. 3.7. Values of $k/100$, $d = \log_{\hat{n}}(q' - (n - \hat{n}))$ and $e = \log_{\hat{n}}(\hat{n} \cdot (\hat{n} - 1)/2)$

Additional test instances

So far we have constructed 9 instances (one for each of the values of s) for each of the 3 datasets – in total 27 instances. Now we expand each of the 27 instances into 5 instances (for

a total number of 135 instances) as follows. Given a set S of frequent itemsets constructed at Step1, we construct five κ -IFM $_{\sigma'}$ instances $(\mathcal{I}, S, \Gamma_{\sigma}, \sigma', size)$ as follows. For each $I \in S$, Γ_{σ} contains the triple $(I, \sigma'_{min}, \sigma'_{max})$, where $\sigma'_{min} = \sigma^{\mathcal{D}}(I) \cdot (1 - \alpha)$, $\sigma'_{max} = \sigma^{\mathcal{D}}(I) \cdot (1 + \alpha)$ and α is one of the values in $\{0.0, 0.05, 0.1, 0.15, 0.2\}$. Thus the basic instances are the one with $\alpha = 0.0$.

We observe that α plays a crucial role. Indeed, for $\alpha = 0$, the support of an itemset in S is constrained to its support in the original dataset. In this case, there is at least one solution, i.e., the original dataset, and the accuracy of the method can be computed by some similarity measure between the obtained dataset and the original one. Instead, for $\alpha > 0$, we constrain the support of each itemset within an interval $[\sigma_{min}, \sigma_{max}]$ with $\sigma_{min} < \sigma^{\mathcal{D}}(I)$, and an itemset may happen to be frequent in the original dataset but not in the synthesized one.

A wide range of experiments were conducted to study the trend of the performance when α varies. We have fixed an upper bound of 0.2 for α , since a deviation larger than 20% w.r.t. the actual support is unlikely to be desirable.

In synthesis, for each dataset \mathcal{D} , for each support s in $\{0.2\%, 0.3\%, \dots, 0.9\%, 1\%\}$, and for each α in $\{0.0, 0.05, 0.1, 0.15, 0.2\}$, we create an κ -IFM $_{\sigma'}$ instance $(\mathcal{I}, S, \Gamma_{\sigma}, \sigma', size)$ where:

- \mathcal{I} is equal to the set of items in the original dataset \mathcal{D} ;
- S is equal to the set of all frequent itemsets according to the threshold s ;
- $\Gamma_{\sigma} = \{(I, \sigma'_{min}, \sigma'_{max}) : I \in S\}$ is built by setting for each itemset $I \in S$ whose support in the original dataset is $\sigma^{\mathcal{D}}$, $\sigma'_{min} = \sigma^{\mathcal{D}} \cdot (1 - \alpha)$ and $\sigma'_{max} = \sigma^{\mathcal{D}} \cdot (1 + \alpha)$;
- $\sigma' = \sigma'_{max} - 4 \cdot 10^{-4} |\mathcal{D}|$, where σ'_{max} is the greatest integer number that divided for the size of \mathcal{D} is smaller than $s \cdot 10^{-2}$;
- $size$ is fixed to $(|\mathcal{D}|, |\mathcal{D}|)$, i.e. $size1$ and $size2$ are set to the number of transactions in the original dataset, indicated as $|\mathcal{D}|$ in Fig. 3.4, e.g., for BMS-View-1 $size1 = size2 = 59602$.

3.4.2 Results

The heuristic column-generation algorithm proposed in 3.3.4 was coded in Java with the Ilog cplex 12.0 library, and the computational experiments were performed on a server with an Intel Xeon E5450 3.0GHz platform, 16GB of RAM, and Linux 64bit as OS. We imposed a time limit of 3 hours to each test.

We assess the effectiveness of the algorithm by a new *accuracy* index defined in the following. Let \mathcal{D}' be the synthesized dataset, \mathcal{D} be the original dataset, and $\mathcal{F}^{\mathcal{D}'}$, $\mathcal{F}^{\mathcal{D}}$ be frequent itemsets extracted from \mathcal{D}' and \mathcal{D} , respectively. The proposed accuracy index is:

$$A(\mathcal{D}') = \frac{1}{|\mathcal{F}^{\mathcal{D}} \cup \mathcal{F}^{\mathcal{D}'}|} \sum_{I \in \mathcal{F}^{\mathcal{D}} \cup \mathcal{F}^{\mathcal{D}'}} 1 - \frac{|\sigma^{\mathcal{D}'}(I) - \sigma^{\mathcal{D}}(I)|}{\max(\sigma^{\mathcal{D}}(I), \sigma^{\mathcal{D}'}(I))} \quad (3.26)$$

By construction, $A(\mathcal{D}')$ values range between 0 and 1, with values close to 1 being rather desirable. Figures 3.8, 3.9, and 3.10 show the accuracy of the algorithm on instances derived from BMS-View-1, from BMS-View-2, and from T10I4D100K, respectively. We now discuss the trend of the accuracy when α varies. It comes as no surprise that accuracy degrades when α increases, and, in fact, the larger a support window, the larger is the gap between the original dataset and the computed solution. However, the figures indicate that very high accuracy measures are obtained in general. Specifically, the accuracy on instances derived from the two real datasets is always greater than 0.89, when $\alpha = 0$. Moreover, it is relevant to notice

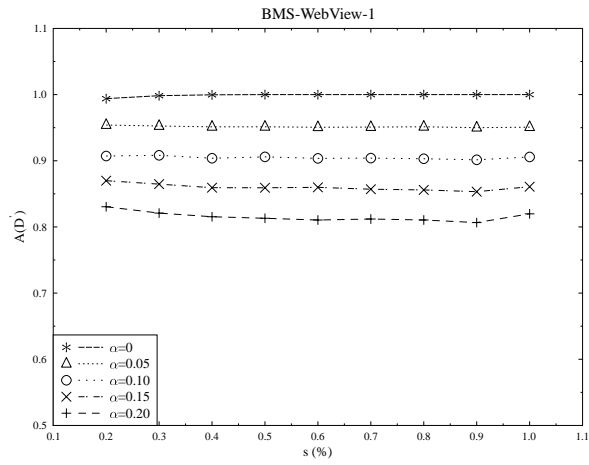


Fig. 3.8. Accuracy on instances derived from the BMS-View-1 dataset.

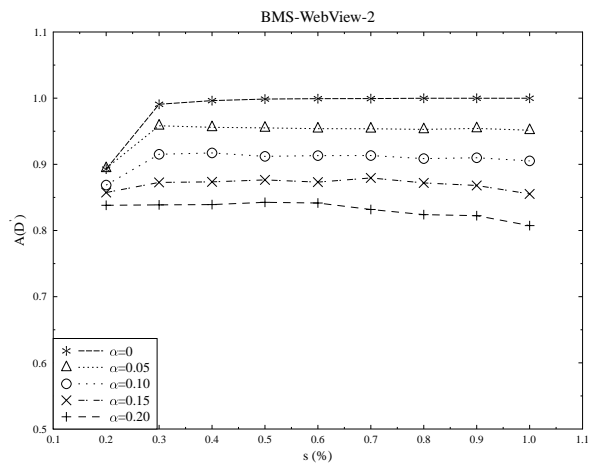


Fig. 3.9. Accuracy on instances derived from the BMS-View-2 dataset.

that the accuracy trend in the artificial dataset T10I4D100K is very different w.r.t. the real datasets. Specifically, the accuracy is always below 0.95 and it seems that the algorithm does

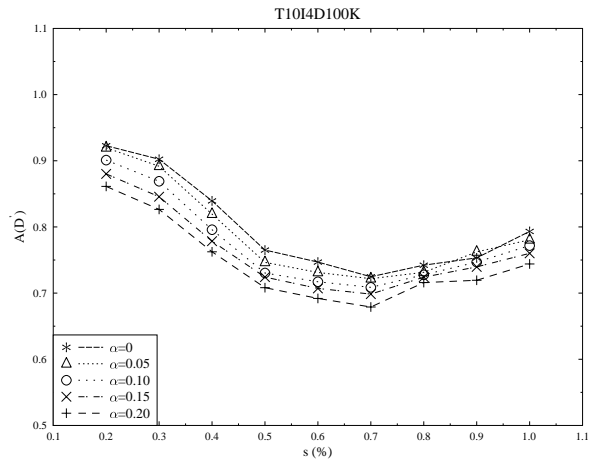


Fig. 3.10. Accuracy on instances derived from the T10I4D100K dataset.

not take any advantage from the type of instance. This is very strange, suggesting the need for researchers to improve the artificial datasets or to use real-world datasets.

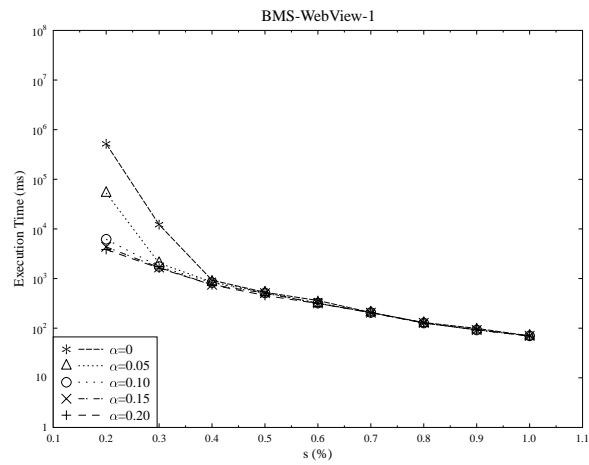


Fig. 3.11. Execution Times on instances derived from BMS-WebView1

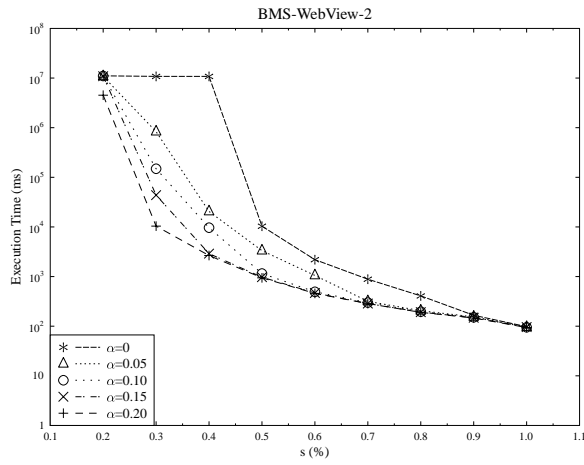


Fig. 3.12. Execution times on instances derived from BMS-WebView2

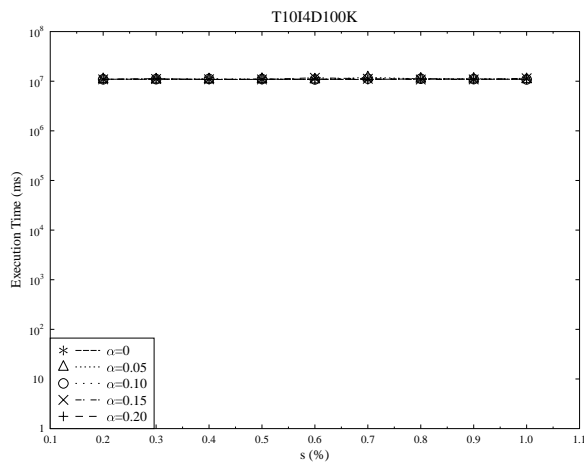


Fig. 3.13. Execution times on instances derived from T10I4D100K

Figures 3.11 – 3.13 report execution times. Differently from the accuracy which improves when α decreases, the execution time worsen with smaller α . We observe that execution time

on the real datasets is affected by α specially for low values of the threshold ($s < 0.5\%$). These results again evidence a completely different trend when the algorithm run on the instances derived from the artificial dataset. In fact, the execution time on instances derived from T10I4D100K is always $10^7 ms$, i.e. the value of the time limit. The encouraging result is that, also on these instances, we achieve a high level of accuracy (always greater than 0.75, when $\alpha = 0$).

The table of average execution times, reported in Fig. 3.14, shows that our method exhibits reasonable execution times (ranging from seconds to a few hours in the worst cases). As expected, a high execution time occurs when there is a large number of constraints in the linear program. To appreciate this, we report in Fig. 3.14 the cardinality of S and Q' , since the number of constraints of the linear program is polynomial in these values. We observe that the method allows us to handle linear programs with an enormous number of variables using a reduced amount of space and time. Limitation on space can be guaranteed in all cases whereas exponential time cannot be ruled out. However, our experiments suggests that exponential time is not encountered in practice, similarly to what happens with the classical simplex algorithm.

s (%)	BMS-WebView-1			BMS-WebView-2		
	$ S $	$ Q' $	time(s)	$ S $	$ Q' $	time(s)
0.2	798	36506	519.090	3683	167688	11084.022
0.3	435	25530	12.216	1340	61083	10811.477
0.4	286	16605	0.928	676	30343	10803.061
0.5	201	11498	0.528	408	17642	10.352
0.6	162	8732	0.361	257	11255	2.192
0.7	133	6260	0.207	187	8614	0.885
0.8	105	4226	0.130	138	6927	0.408
0.9	90	3260	0.099	113	6051	0.166
1.0	77	2633	0.070	81	4827	0.98

Fig. 3.14. Average execution times and cardinality of S and Q' on instances derived from the two real datasets.

In the third series of figures we present results for a compactness measure, dt , defined as the ratio between the number of different transactions of the synthesized dataset, $trans(\mathcal{D}')$, and the number of different transactions of the original dataset, $trans(\mathcal{D})$:

$$dt = \frac{trans(\mathcal{D}')}{trans(\mathcal{D})}.$$

Figures 3.15, 3.16 and 3.17 show values of dt for instances derived from BMS-View-1, from BMS-View-2, and from T10I4D100K, respectively. Note that only for the artificial dataset with threshold values of 0.8%, 0.9%, 1%, the dt value is high. Instead, for other cases the number of different transactions in the synthesized dataset is much less than in the original dataset. This means that our method yields datasets more compact than real ones.

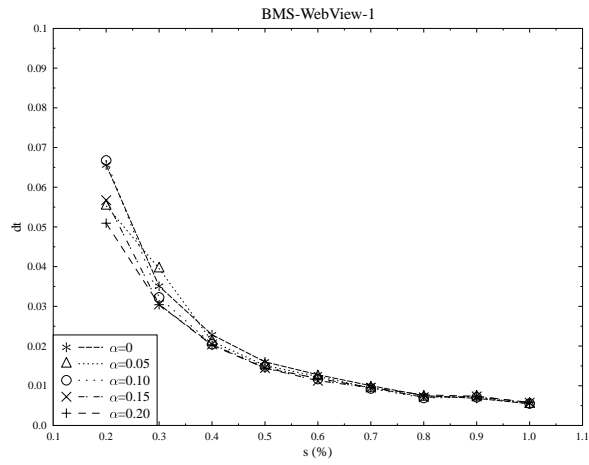


Fig. 3.15. Compactness measure for instances derived from BMS-View-1.

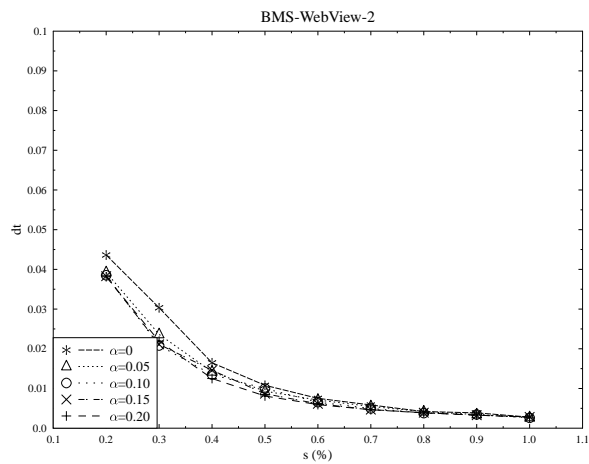


Fig. 3.16. Compactness measure for instances derived from BMS-View-2.

3.5 Related Works

IFM has been firstly introduced in [Mie03], where the problem was analyzed from the computational complexity point of view, by showing that deciding whether there is a dataset com-

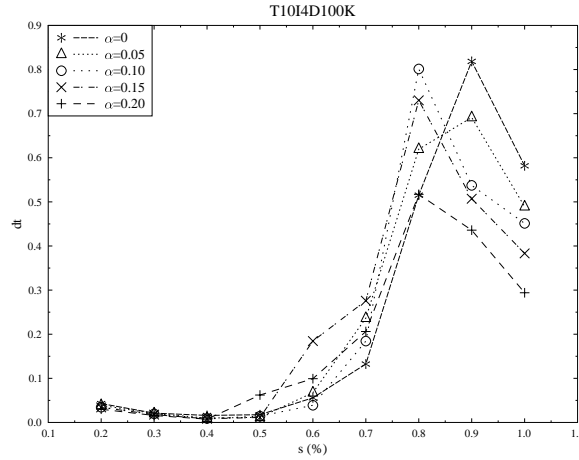


Fig. 3.17. Compactness measure for instances derived from T10I4D100K.

patible with the given frequent sets is NP-hard. Differently from our research, the lower and upper bounds of support in the formulation were assumed to coincide and no restriction was enforced on the itemsets which do not belong to the set S – such assumptions may result to be too simplistic in some case, as already discussed in the introduction.

This problem was subsequently considered by Calders, who introduced the $FREQSAT$ perspective [Cal04, Cal07, CG07, Cal08] where the support of the itemsets is measured in terms of their frequency. Under this novel formulation, several variants of $FREQSAT$ have been studied and a complete parameterizations of their intrinsic difficulty was provided. For example, in $FREQSAT_{\{NTRANS\}}$ the dimension of the database is fixed to $NTRANS$ value and it is proved that this problem is in PSPACE and NP-hard; in $FREQSAT_{\{NDUP\}}$ each transaction in the database can appear at most $NDUP$ times and it is proved that this problem is in PSPACE and \mathcal{PP} -hard; finally in $FREQSAT_{\{NTRANS,NDUP\}}$ both the constraints of $FREQSAT_{\{NTRANS\}}$ and $FREQSAT_{\{NDUP\}}$ hold and it is proved that this problem has the same complexity of $FREQSAT_{\{NTRANS\}}$.

Observe that $NDUP$ constraints introduce a duplicate threshold also for itemsets not in S but, unfortunately, such a threshold is the same for all transactions and, then, also for the itemsets in S , that should instead have larger bounds. Furthermore, there is no limitations on the supports for itemsets not in S . This is a serious drawback, because permits that “unwished” frequent itemsets may arise even in presence of an upper bound on the number of duplicates. Our approach fix a maximum support for all itemsets not in S thus avoiding this possibility.

It is worth noting that the classical IFM formulation introduced in [Mie03] can be specialized in $FREQSAT$ problems for the case with a constraint on the database size, since supports can be transformed in frequencies and vice versa.

In the light of the intractability of the inverse frequent itemset mining problem, approximation strategies have been discussed in [WW05]. In particular, the authors introduce the $APPROSUPPORT$ problem, where they asked whether it is possible to satisfy the various support

constraints in an approximate fashion. The approach adopted here is a ILP formulation of APPROXSUPP, and a relaxed version of the integer formulation is used with heuristic strategies to generate approximate transaction database, by providing rounding methods to get an integer solution. However such formulation uses binary variables, meaning that an item belong or not to a transaction in the resulting database solution, thus the main drawback in this approach is that the relaxed version of the formulation introduces strong approximations that should compromising the reliability of the results.

A heuristic approach to generate a database satisfying the given frequency constraints was firstly proposed in [WWWL05]. The approach founds on the Iterative Proportional Fitting (IPF) method to estimate contingency tables for each itemsets, and on graphical decomposition techniques to decompose items into independent components in order to reduce the exponential explosion of such a tables and then apply IPF algorithms only on the irreducible components. As a consequence, the feasibility of this approach depends on the assumption that many of the items are (conditionally) independent.

Another heuristic approach was discussed in [RMZ03], where a method to generate basket datasets for benchmarking activities is discussed which is applicable when the length distributions of frequent and maximal frequent itemset collections is available. Unfortunately, as discussed in [CO05], even though the generated synthetic data set preserves the length distributions of frequent patterns, the problem is that the size of transactions generated usually is much larger than that of the original database while the number of items generated is much smaller.

In a previous work [GSS09b], we have investigated the IFM_S problem obtained by considering a minimum and a maximum support constraint over each itemset in S , while itemsets not in S have support zero. Even though the size of the output database is not fixed in the formulation, such size can be imposed by further constraining the support of itemsets in S . Moreover, within this setting, we proposed an algorithm that always satisfies the maximum support constraints, but which treats minimum support constraints as soft ones that are enforced as long as it possible. This algorithm has inspired us in designing the heuristic column-generation algorithm.

3.5.1 Solving FREQSAT with Heuristic Column Generation Algorithm

We observe that FREQSAT can be directly formulated as an LP problem using our notation as follows:

$$\text{minimize } \sum_{i=1}^{2m+1} v_i \quad (3.27)$$

subject to

$$v_{Q(i)} + \sum_{j \in T} a_{ij} x_j \geq l_{Q(i)} \quad i \in Q \quad (3.28)$$

$$v_{m+Q(i)} - \sum_{j \in T} a_{ij} x_j \geq -u_{Q(i)} \quad i \in Q \quad (3.29)$$

$$v_{2 \cdot m+1} + \sum_{j \in T} x_j = 1 \quad (3.30)$$

$$v_i \in \mathbb{Q}_0 \quad 1 \leq i \leq 2m+1 \quad (3.31)$$

$$x_i \in \mathbb{Q}_0 \quad i \in T. \quad (3.32)$$

For each $i \in V$ (thus, for each itemset I_i in S), the variable x_i represents the frequency of I_i as a rational number so that the usage of integer linear programming is not anymore necessary. The number of constraints as well as of artificial variables is strongly reduced as there is no restriction for the itemsets non in S . The constraint (3.30) imposes that the sum of all frequencies be 1. Obviously this problem can be effectively solved using the heuristic column-generation algorithm. In the conclusion we shall discuss some experiments we have already done in such an application of the algorithm.

3.5.2 Comparison with IPF

In this subsection, we compare our method with the IPF method proposed in [WWWL05]. The method exploits similarity measures between original frequent itemsets ($\mathcal{F}^{\mathcal{D}}$) and those occurring in the output dataset ($\mathcal{F}^{\mathcal{D}'}$). To this aims, the authors in [WWWL05] compute the *Jaccard*, *Dice*, and *Overlap* indices (see [HK06] chapter 5, for general background and discussion of similarity measures):

$$\begin{aligned} Jaccard(\mathcal{F}^{\mathcal{D}}, \mathcal{F}^{\mathcal{D}'}) &= \frac{|\mathcal{F}^{\mathcal{D}} \cap \mathcal{F}^{\mathcal{D}'}|}{|\mathcal{F}^{\mathcal{D}} \cup \mathcal{F}^{\mathcal{D}'}|} \\ Dice(\mathcal{F}^{\mathcal{D}}, \mathcal{F}^{\mathcal{D}'}) &= \frac{2 \cdot |\mathcal{F}^{\mathcal{D}} \cap \mathcal{F}^{\mathcal{D}'}|}{|\mathcal{F}^{\mathcal{D}}| + |\mathcal{F}^{\mathcal{D}'}|} \\ Overlap(\mathcal{F}^{\mathcal{D}}, \mathcal{F}^{\mathcal{D}'}) &= \frac{|\mathcal{F}^{\mathcal{D}} \cap \mathcal{F}^{\mathcal{D}'}|}{\min(|\mathcal{F}^{\mathcal{D}}|, |\mathcal{F}^{\mathcal{D}'}|)} \end{aligned}$$

As in [WWWL05], we applied our κ -IFM $_{\sigma'}$ method on two instances generated with (\mathcal{D} =BMS-View1, $s = 0.6$, $\alpha = 0$), and (\mathcal{D} =BMS-View2, $s = 0.7$, $\alpha = 0$), respectively. Results on these instances are reported in Fig. 3.18. Please note that a similarity index is computed only when the support threshold value used for mining is greater than or equal to the support threshold used for data generation. On the other hand, when support threshold values used for mining (the column indicated by s (%) in the table) are smaller than that used for data generation ($s = 0.6$ and $s = 0.7$), statistics on original dataset are not available.

BMS-Web	s (%)	Jaccard		Dice		Overlap	
		κ -IFM $_{\sigma'}$	IPF	κ -IFM $_{\sigma'}$	IPF	κ -IFM $_{\sigma'}$	IPF
View-1 s=0.7	0.7	1.000	0.940	1.000	0.969	1.000	0.992
	0.8	1.000	0.883	1.000	0.938	1.000	0.934
	0.9	1.000	0.893	1.000	0.944	1.000	0.954
	1.0	1.000	0.887	1.000	0.940	1.000	0.959
View-2 s=0.6	0.6	1.000	0.696	1.000	0.768	1.000	1.000
	0.7	1.000	0.708	1.000	0.739	1.000	0.964
	0.8	1.000	0.710	1.000	0.830	1.000	0.928
	0.9	0.991	0.722	0.995	0.838	1.000	0.976
	1.0	1.000	0.701	1.000	0.824	1.000	0.910

Fig. 3.18. Comparison of κ -IFM $_{\sigma'}$ accuracy vs. IPF method

This comparison highlights that very high similarity measures are obtained by our method, outperforming those obtained by IPF. Even though the *Jaccard*, *Dice*, and *Overlap* indices are the most common way to formalize the concept of similarity, the accuracy index that we have proposed, see equation (3.26), provides a more fine-grained evaluation of the effectiveness

of the inverse frequent mining. In fact, these three indices assess similarity of two transactional datasets by looking for the set of frequent itemsets, but do not take into account the frequency values of these itemsets. In our opinion, ignoring the number of occurrences of frequent itemsets, i.e. their support, does not properly evaluate similarity of two transactional datasets (especially when at least one of them is of a large size). On the contrary, our accuracy measure looks inside the frequent itemset, and strongly penalizes the mismatches among the number of occurrences of such itemsets in the two datasets.

3.6 Applications of IFM

To better appreciate the practical relevance of the results of this chapter, we stress that the IFM problem arises in a number of novel applications. We discuss a few of them below.

Database generation method for benchmarking

Consider a scenario in which a data set is needed as an input of data mining applications, for analysis purposes or simply for testing the implementation of new reasoning methods. The unavailability of real data often exacerbates in the case of lowly structured multi-organization virtual enterprises where the partners will never agree to share their own information, or no tracking infrastructure exists for collecting and storing log data. In fact, such a situation is likely to happen in many real-world application scenarios falling in the contexts of industrial districts and supply chains, as we experienced in the industrial research project "TOCAI: Knowledge-oriented technologies for enterprise aggregation in Internet". A major problem that we have faced in this project is the scarcity and/or low quality of real log data, mainly due to privacy reasons or to commercial strategies. On the other hand, the usage of artificial data sets in place of those reals has demonstrated impracticable since the former may not reflect the actual properties of real processes, and in fact there have been extensive study [ZKM01, Jea05, HK06, TF09] proving that the artificial data sets have very different characteristics from the real-world data sets and as consequence, the results of the mining algorithms are very different when obtained over syntectic datasets instead of the real ones. This is also true for frequent itemset mining as demonstrated in [ZKM01, HCXY07]. In order to ensure high quality experimental findings in TOCAI, we have applied an inverse set mining technique for generating benchmark datasets with features mined from real-world datasets. Specifically, we have used frequent itemsets together with their supports provided by the organizations involved in TOCAI as the features. As result we have devised a privacy aware database generation method, since the high complexity of the inverse frequent set mining problem somewhat guarantees that it is very hard to break the privacy. This argumentations is first discussed in [WWWL05] as motivating example.

Privacy preserving data publishing

As a further motivating application scenario for IFM, assume you are the owner of a transaction database which is willing to make available its data to any interested party, but you are wondered how to keep private information or sensitive knowledge from being disclosed. This is also the scenario depicted in a lot of contributions in the privacy preserving data publishing field [DN03, AS00, ESAG02, AA01, JA05, WWWL05, ABE⁺99, SVC01, RH02, JX08], where approaches proposed in the last decade are mainly devoted to modify the original

database before publishing it by means of various sanitization methods, such as anonymization, hiding, perturbation. Notably, the drawback of these approaches is that modifying the original database to limit the disclosure of sensitive knowledge may impact on other, non-sensitive, knowledge [EGS03, LKT08, OZ03]. For example, in [JA05] a downside of anonymizing transaction database is that the data mining results computed from such database are not always very informative. When you decide to hide association rule based data sanitization techniques, as in [ESAG02], you cannot control the hiding effects of confidential rules obviously, which can only be validated after sanitization.

Conversely, recent literature has proposed a solution to the data sanitization problem which is inspired to inverse frequent set mining. Even though in a preliminary form, this idea is first depicted in [COL04], where a data reconstruction algorithm for hiding sensitive frequent itemsets is based on a coarse Constraint-based Inverse Itemset Lattice Mining procedure (CIILM). Inspired from this intuition are the works in [NLO06, GTTY06], where the inverse mining method is the basic technique for data reconstruction while hiding association rule.

Condensed and succinct representations of databases

Another possible application is the construction of condensed representations [CRfB05] of database. Condensed representation of a transactional database is a minimal representation of dataset storing a non-derivable cover of all frequent itemsets, in place of the complete collection of all frequent itemsets. The problem here is to define scalable methods discovering itemsets for which the support can be derived, so they can be removed without losing information, thus resulting in the so called Non-Derivable Itemsets representation. In [CG07, Cal08, CG03] it is argued that inverse frequent mining methods can be an interesting building block solution to the question of what can be derived from some given frequencies and can be used to see whether the frequency of a certain itemset in a collection is uniquely determined by the other itemsets in that collection. This is also the approach applied in [CG07].

We note that a classical database scheme together with a number of integrity constraints can be thought of as a condensed ("succinct") representation of all admissible database instances. In analogy with this point of view, IFM can be seen as transactional database scheme where the integrity constraints are represented by support thresholds. The IFM decision problem then consists in verifying whether the scheme admits instances and a search solution returns one of them. Within this more general framework the IFM problem can be considered under a new perspective and we are currently conducting some research in this direction. In particular, as an extension of support constraints, we have introduced count constraints that require the results of given count operations on a relation to be within a certain range, and we have formulated a new decisional problem, the *Inverse OLAP*: given a schema (that typically represents a star schema of an OLAP application), does there exist a relation instance satisfying a set of given count constraints? We have shown that IFM is as a special case of inverse OLAP.

Even though this chapter will not address all these issues, we believe the research problem formulated and addressed in this work takes a solid step in this direction, and particularly sheds light on a list of important challenges related to devising effective procedure to solve the inverse mining of transactional databases.

Count Constraints and the Inverse OLAP Problem

A typical problem in database theory is to verify whether there exists a relation (or database) instance satisfying a number of given dependency constraints. This problem has recently received a renewed deal of interest within the context of data exchange, but the issue of handling constraints on aggregate data has not been much investigated so far, notwithstanding the relevance of aggregate operations in exchange systems. This chapter introduces count constraints that require the results of given count operations on a relation to be within a certain range. Count constraints are defined by a suitable extension of first order predicate calculus, based on set terms, and they are then used in a new decisional problem, the *Inverse OLAP*: given a schema (that typically represents a star schema of an OLAP application), does there exist a relation instance satisfying a set of given count constraints? The new problem turns out to be NEXP complete under various conditions: program complexity (i.e., number of attributes in the relation scheme and the size of constraints), data complexity (i.e., size of attribute domains) and combined complexity. We illustrate how count constraints can be used into a data exchange system context, where data from the source database are transferred to the target database using aggregate operations.

4.1 Introduction

The problem of inverse frequent set mining (IFM) [Mie03, Cal04, Cal07], consists in verifying whether there exists a transaction database D satisfying a given set Γ_S of support constraints on some itemsets, that are typically the frequent ones

The original aim of our research was mainly to present a general formulation of the IFM problem in which support constraints on unfrequent itemsets are given in a succinct and flexible way using generic properties of them. We eventually realized that the IFM problem represents the succinct description of an itemset dataset in the same way as the integrity constraints of a relational database schema are a succinct description of all satisfying databases. The search for a general framework stimulated us to consider the problem under a new perspective: as a special case of constraint satisfaction for an OLAP star schema. Indeed IFM can be formulated as a particular problem of integrity constraint satisfaction on a relational scheme $R(T, I)$ with two attributes T (the ID of a transaction) and I (the ID of an item) with domain \mathcal{I} . Given a relation r on R , $G = Group(r)By(T)$ divides r into a number of groups, one for each transaction ID. Then, for each $K \subseteq \mathcal{I}$, $G^\sigma(K) = G_{(I \supseteq K)}$ filters the groups in G for which K is a subset of the projection on I .

We then consider a new type of integrity constraint for R , called *count constraint*, requiring that, for each relation r and for each $K \subseteq \mathcal{I}$, the number of filtered groups in $G^\sigma(K)$ be within a given range — thus enforcing support restrictions. More in general count constraints prescribe that the results of given count operations on a relation to be within a certain range. To enable an expressive description of count constraints, we adopt a simple logic formalism with set terms and count aggregates. This formalism is similar to the one used in the data exchange setting [FKP05] and allows us to express count constraints on a star schema, used in OLAP analysis, which is characterized by multidimensional data cubes that enable manipulation and analysis of data stored in a source database from multiple perspectives in a fast way [CD97, LL03, Han05]. Each *dimension* of a multidimensional data cube stores the values of an attribute and each cell contains an aggregate value, called *measure* (e.g., count), based on aggregation criteria that depend on coordinates of this cell in the data cube. As tangible proof of the capabilities of our logic-based language to enforce count constraints on a star schema, we illustrate its usage in a "non-toy" example (not another employee - department example variation) that, in our believe, witnesses the potential relevance of our proposal.

Count constraints are later used to define a new decisional problem, the *Inverse OLAP*: given a schema (that typically represents a star schema of an OLAP application), does there exist a relation instance satisfying a set of given count constraints? Inverse OLAP is relevant for the generation of synthetic data cubes having the same characteristics of real-world ones in terms of aggregation patterns. The formulation of this problem shows a strict relationship between classical integrity constraint satisfaction on database schemes and inverse mining problems: both problems provide a succinct description of satisfying instances. We prove that the new problem is NEXP-complete under various conditions: combined complexity (i.e., both attribute number, size of count constraints and domain sizes are part of the input), program complexity (i.e., the domain sizes are constant) and data complexity (i.e., the number of attributes and the size of constraints are constant). This result is in our view important for three reasons. The first one is that it precisely characterizes the complexity of the problem. The second reason is that it shows an exemplary of NEXP-complete problem different from classical ones that are deterministic succinct representations of NP-complete problems, deterministic in the sense that there is a unique expanded description although it is generally computed in exponential time; instead the General Inverse OLAP problem is a non-deterministic succinct representation in the sense that expansion process requires non deterministic exponential time.

The third reason is that Inverse OLAP can be easily extended to provide a new setting for data exchange that is capable to handle migration of aggregate data. We show a first step towards this direction, that for the moment consists just in the illustration of two meaningful examples: privacy preserving data migration and datacube construction. Note that privacy preserving sharing of data is an important motivation for Inverse OLAP: organizations may be interested in exchanging some patterns of their own data but not actual raw data.

The chapter is organized as follows. In Section 2 we present some useful background to our research. We start from some basic notions on inverse frequent set mining problem, that can be considered the ancestor of Inverse OLAP, then review some basic notions on constraint satisfaction on OLAP and relational database relations and finally overview the main issues of data exchange setting. In Section 3 we introduce the logic language for describing count constraints and illustrate their usage in a motivating example in Section 3.1. In Section 4 we present the formulation of the Inverse OLAP and prove that it is in NEXP. To prove completeness, we introduce in Section 4.1 a program-complexity version of it, called *Binary Domain Inverse OLAP*, that is proved to be NEXP-complete using a reduction from succinct Hamiltonian circuit. Then in Section 4.2 we define another simple, complementary specialization of Inverse OLAP, called *Binary Attribute Inverse OLAP*, that is a sort of rewriting of the IFM

problem. Binary Attribute Inverse OLAP is thereafter used to prove that also the data complexity of Inverse OLAP is NEXP-complete. In Section 5 we illustrate how count constraints can be used into a data exchange system context, where data from the source database are transferred to the target database using aggregate operations. Finally in Section 6 we draw the conclusion and argue that, following the approach used in a recent paper for finding efficient solutions to IFM, Inverse OLAP could be solved in a reasonable amount of time also for a large input size.

4.2 Preliminaries and related work

The main problems we deal with in this chapter (count constraints and Inverse OLAP) are rather novel and, at the best of our knowledge, there is not much direct related work. Nevertheless, there is an important connection with another inverse problem, called frequent itemset mining - actually this problem has inspired the initial lines of our research. In perspective, we also believe that our work could have some interesting relationships with another important problem: data exchange systems.

4.2.1 From IFM to Inverse OLAP

The starting point for generalizing IFM in a more general context is that it can be formulated as a particular problem of integrity constraint satisfaction on a relational scheme $R(T, I)$ with two attributes T (the ID of a transaction) and I (the ID of an item). Let \mathcal{T} and \mathcal{I} be the domains on T and I respectively. Given a relation $r \subseteq \mathcal{T} \times \mathcal{I}$ on R , $G = \text{Group}(r)\text{By}(T)$ divides r into a number of groups, one for each transaction ID. Then, for each $K \subseteq \mathcal{I}$, $G^\sigma(K) = G_{\text{Having}(I \supseteq K)}$ filters the groups in G for which K is a subset of the projection on I .

Consider now a new type of integrity constraint for R , called *count constraint*, requiring that, for each relation r and for each $K \subseteq \mathcal{I}$, the number of filtered groups in $G^\sigma(K)$ be within a given range — thus enforcing support restrictions. Under this perspective, IFM becomes a particular case of the problem of integrity constraint satisfaction for a relation scheme: does there exist a relation r on R such that r satisfy all count constraints?

To strengthen the relationship between inverse mining problem and relation integrity constraint satisfaction, in the next section we further elaborate on the structure of the relation $R(T, I)$ and count constraints. This relation is a simple case of *star schema*, that is a relation scheme whose attributes are measures and (possibly layered) dimensions. A star schema is used in OLAP analysis, which is characterized by multidimensional data cubes that enable manipulating and analyzing data stored in a source database from multiple perspectives in a fast way [CD97, LL03, Han05]. Each *dimension* of a multidimensional data cube stores the values of an attribute and each cell contains an aggregate value, called *measure* (e.g., count), based on aggregation criteria that depend on coordinates of this cell in the data cube.

A data cube is defined by designing a relational database schema as a multidimensional view on the source database, typically a star schema representing a fact table, where both measures and dimensions are stored. After the definition of the star schema, the data cube is built by performing a number of Projection/Selection/GroupBy queries over the source database. In case of two dimensions A and B for which the functional dependency $A \rightarrow B$ holds (*layered dimensions*), the dimension B can be removed from the fact table and stored into an additional ad-hoc (dimension) table $D(A, B)$ having A as key, thus obtaining a so called

snowflake schema; in this case an inclusion dependency is added which enforces that all values of the dimension A in the fact table be contained in the corresponding column of the dimension table.

In the next sections we shall define count constraints for a star schema and define a new inverse problem, called *Inverse OLAP*: given a star schema $R(A_1, \dots, A_n)$ and a number of count constraints, does there exist a fact table (i.e., a relation) on R satisfying the integrity constraints?

The problem of deciding the existence of a database satisfying a given set of integrity constraints is tightly related with the implication problem of integrity constraints and with the problems of query answering and containment under inclusion dependencies. Many decidability results in these settings have been established for classes of constraints which admit a finite *chase* [BV90], i.e., a finite “canonical” database, witnessing the satisfiability of the constraints at hand. In particular, most of such classical studies in database theory focused on inclusion dependencies and functional dependencies [ZO97, Ros06, CKV90].

4.2.2 Data Exchange

As we shall illustrate later in the chapter, Inverse OLAP may have strict relationships with the data exchange problem, first defined in [FKP05]. This is the problem of migrating a data instance from a source schema to a target schema such that the materialized data on the target schema satisfies the integrity constraints specified by it. Data exchange is different to data integration [Len02] because the data is indeed materialized at the target schema, which is not always the case for data integration settings. The mapping of the data from the source to the target schema is given by source- to-target TGDs (Tuple Generating Dependencies). Additionally, the target schema specifies target constraints in form of EGDs (Equality Generating Dependencies) and TGDs, which the imported data must satisfy. Fagin et al. [FKP05] introduced the notion of universal solutions, and argued that these should be the preferred solutions to materialize in data exchange (i.e. a universal solution can be mapped through a homomorphism into any other solution). Moreover, it was proved in that, the answer of a conjunctive query on a universal solution, is contained in all answers of the same query on each mapping solution. Other important results on data exchange can be found in [FKP05, GLLR07, GN08, FKPT09, APR11]. Aggregate queries in the data exchange setting has been investigated in [AK08].

4.3 Count Constraints

Let $U = \{A_1, \dots, A_n\}$ be a set of attributes on the domains D_1, \dots, D_n with given cardinalities d_1, \dots, d_n . We denote $\cup_{i=1,n} D_i$ by \mathbf{D} and $\sum_{i=1,n} d_i$ by \mathbf{d} . Given a subset $L = \{A_{i_1}, \dots, A_{i_k}\}$ of $k \geq 0$ attributes in U , a *tuple* a on L is a mapping from $A_{i_1} \times \dots \times A_{i_k}$ to D_{i_1}, \dots, D_{i_k} — assuming an implicit ordering of the attributes in L , we represent a simply as a tuple $[a_1, \dots, a_k]$ such that for each j , $1 \leq j \leq k$, $a_j \in D_{i_j}$. We denote the set of all tuples on L by D^L ; a subset (not necessarily proper) of D^L is called a *table* on L . If $k = 0$ then D^L only contains the empty tuple. In addition, given $k > 0$, $[a_1, \dots, a_k]$ with $a_1, \dots, a_k \in \mathbf{D}$ will be called an *untyped tuple* and we shall write that $[a_1, \dots, a_k] \in \mathbf{D}^k$.

A *relation scheme* is a pair consisting of a relation name and a list of attributes. In this chapter we shall only deal with exactly one relation scheme containing all attributes in

U , say $\mathcal{R}(A_1, \dots, A_n)$. We also assume that the domains of the attributes D_1, \dots, D_n are stored in suitable tables of mono-attribute relation schemes — we call such relation schemes $\mathcal{D}_1, \dots, \mathcal{D}_n$. The size of the description of \mathcal{R} and the involved domains is then \mathbf{d} and a *relation* on \mathcal{R} (also called an *instance* of \mathcal{R}) is any table on U . \mathcal{R} represents a typical *star schema* whose dimensions are stored into a unique *fact table*, that is a relation on \mathcal{R} - as we are only interested to count aggregation, we omit to include measures. Some of the dimensions could be organized in layers by means of Functional Dependencies (FDs) — for instance the FDs $A \rightarrow B$ and $B \rightarrow C$ state that the values of dimension A are grouped at a first level B and at a second level C . In correspondence of FDs we may have additional domain relations describing hierarchies among two dimensions, e.g., $\mathcal{D}_{A,B}$ and $\mathcal{D}_{B,C}$.

We next introduce an extension of first order predicate calculus to define count constraints on the instances of \mathcal{R} . The predicate symbols only are \mathcal{R} , the domain relation schemes $\mathcal{D}_1, \dots, \mathcal{D}_n$ and possible dimension hierarchy domains. The *constants* of the language are the domain values (*domain constants*) and all (non-negative) integers.

Besides to the constants, the Herbrand universe includes constant set terms defined as follows: given any subset S of U , a constant set term is a set of tuples (i.e., a table) over S . To avoid to have a super-exponential number of set terms in the universe, we fix a maximum size \hat{k} for the arity of S , thus \hat{k} is a (typically small) constant given as part of the input of our problem — this is not an actual limitation in practical applications for which the value of \hat{k} is often 1. Let $\mathbf{H}_{\hat{k}}$ denote the Herbrand universe; furthermore, given $p \geq 0$, $\mathbf{H}_{\hat{k}}^p$ denotes the set of all p -tuples on the the Herbrand universe.

Given the attributes A and B with domains $\{a_1, a_2, a_3\}$ and $\{b_1, b_2\}$ respectively, examples of constant set terms on $\{A, B\}$ are $\{[a_1, b_1], [a_2, b_1], [a_3, b_2]\}$ and $\{[a_2, b_1]\}$, while $\{[a_1], [a_3]\}$ and $\{[a_2]\}$ are constant set terms on $\{A\}$. There is an interpreted function symbol *count* (denoted by $\#$) that can be applied to a set term T to return the number of tuples in T (i.e., the cardinality of the table represented by T).

Our language is equipped with a countable number of variables (denoted by capital letters) and makes use of the following types of terms:

- *simple term*: it is either a domain constant or a variable;
- *set term*: it is either a constant set term or a *formula term*, defined as $\{X_1, \dots, X_s : \alpha\}$, where X_1, \dots, X_s are variables, $s \leq k$ and α is a *count formula*, defined next, in which X_1, \dots, X_s occur as free variables (similar notation for set terms and aggregate predicates has been used in the dlw system [FPL⁺08]);
- *integer term*: it can be either an integer or a function term $\#(T)$, where T is either a variable or a set term.

An *atom* can have one of the following formats:

- $\mathcal{R}(t_1, \dots, t_n)$, where t_1, \dots, t_n are simple terms (*relation predicate*);
- $\mathcal{D}_i(t)$, $\forall 1 \leq i \leq n$, where t is a simple term, and for each dimension hierarchy domain on a pair of attributes A_i and A_j , $\mathcal{D}_{A_i, A_j}(t_1, t_2)$, where t_1 and t_2 are simple terms (*domain predicates*);
- *comparison predicates* of the following types:
 - $t_1 = t_2$ and $t_1 \neq t_2$, where t_1 and t_2 are terms (*equality predicate* and *disequality predicate*, respectively);
 - $t_1 < t_2$ and $t_1 \leq t_2$, where t_1 and t_2 are either variables or integer terms (*less predicates*) — we shall use $t_1 \leq t_2 \leq t_3$ as a shorthand for $t_1 \leq t_2 \wedge t_2 \leq t_3$;
 - $t_1 \subseteq t_2$, $t_1 \subset t_2$ and $t_1 \not\subseteq t_2$, where t_1 and t_2 are variables or set terms (*subset predicate*, *strict subset predicate* and *non-subset predicate*, respectively).

A *count constraint* C is a formula of type:

$$\forall \mathbf{X} (\alpha \rightarrow \beta_{min} \leq \#(\{ \mathbf{Y} : \gamma \}) \leq \beta_{max})$$

where:

- \mathbf{X} and \mathbf{Y} are disjunct lists of variables - \mathbf{X} can be empty;
- β_{min} and β_{max} are integers, but β_{max} can be set to ∞ to denote that the right hand side count predicate simply reduces to $\beta_{min} \leq \#(\{ \mathbf{Y} : \alpha \})$;
- α is a (possibly empty) conjunction of domain and comparison predicates, whose variables are in \mathbf{X} — a possible non-constant set term occurring in α must be of the form $\{ \mathbf{W} : \delta \}$, where δ is a conjunction of domain predicates and the variables in \mathbf{W} are distinct from the ones in \mathbf{X} and all occur in δ ;
- every variable $X \in \mathbf{X}$ must be *bound* in α , that is:
 - X occurs in a domain predicate, or
 - X occurs in a comparison predicate $X = t$, $t = X$, $X \subset t$ or $X \subseteq t$ such that t is a constant or a bounded variable or a set term $\{ \mathbf{W} : \delta \}$ in which all the variables in δ are bound;
- γ can be either
 - i $\forall \mathbf{Z} \mathcal{R}(t_1, \dots, t_n)$, where \mathbf{Z} is a possibly empty list of variables, distinct from the ones in \mathbf{X} and \mathbf{Y} , t_1, \dots, t_n are simple terms, all variables in $\mathbf{Y} \cup \mathbf{Z}$ occur as terms in \mathcal{R} and all variables in \mathcal{R} are in $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$ (*tuple count constraint*), or
 - ii $t * \{ \mathbf{Z}_1 : \forall \mathbf{Z}_2 \mathcal{R}(t_1, \dots, t_n) \}$, where $*$ can be $=$ or \subset or \subseteq , t is either a constant set term or a variable in \mathbf{X} , \mathbf{Z}_1 and \mathbf{Z}_2 are disjunct lists of variables distinct from the ones in \mathbf{X} and \mathbf{Y} (\mathbf{Z}_2 can be empty), t_1, \dots, t_n are simple terms, all variables in $\mathbf{Y} \cup \mathbf{Z}_1 \cup \mathbf{Z}_2$ occur as terms in \mathcal{R} and all variables in \mathcal{R} are in $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}_1 \cup \mathbf{Z}_2$ (*group count constraint*).

Given a relation scheme \mathcal{R} together with its domain relations and a relation r on \mathcal{R} , a constraint C as above is evaluated on r as follows:

- a relation (resp., domain) predicate with all constant terms is true if the corresponding tuple is in r (resp., the domain relation) and false otherwise;
- a comparison predicate $t_1 * t_2$ is evaluated to true if the evaluation function $eval$ is defined for both t_1 and t_2 and $eval(t_1) * eval(t_2)$ ¹ or to false otherwise - for each term t , $eval$ is defined as follows:
 - if t is a constant then $eval(t) = t$;
 - if $t = \#(t')$ and $eval(t')$ is a set s then $eval(t) = |s|$ (i.e., cardinality of s);
 - if $t = \{ \mathbf{W} : \delta[\mathbf{W}] \}$, say with $|\mathbf{W}| = p > 0$, and the variables occurring in δ are only those in \mathbf{W} , then $eval(t) = \{ \mathbf{w} \in \mathbf{D}^p \mid \delta[\mathbf{W}/\mathbf{w}] \text{ is evaluated to true} \}$;
 - if $t = \{ \mathbf{Y} : \forall \mathbf{Z} \mathcal{R}(t_1, \dots, t_n)[\mathbf{Y}, \mathbf{Z}] \}$, say with $|\mathbf{Y}| = p > 0$ and $|\mathbf{Z}| = q \geq 0$, and the variables in \mathcal{R} are exactly those in $\mathbf{Y} \cup \mathbf{Z}$, then $eval(t) = \bigcup_{\mathbf{z} \in \mathbf{D}^q} \{ \mathbf{y} \in \mathbf{D}^p \mid \mathcal{R}(t_1, \dots, t_n)[\mathbf{Y}/\mathbf{y}, \mathbf{Z}/\mathbf{z}] \text{ is evaluated to true} \}$;
 - if $t = \{ \mathbf{Y} : s * \{ \mathbf{Z}_1 : \forall \mathbf{Z}_2 \mathcal{R}(t_1, \dots, t_n)[\mathbf{Y}, \mathbf{Z}_1, \mathbf{Z}_2] \} \}$, say with $|\mathbf{Y}| = p > 0$, $|\mathbf{Z}_1| > 0$ and $|\mathbf{Z}_2| \geq 0$, s is a constant, and the variables in \mathcal{R} are exactly those in $\mathbf{Y} \cup \mathbf{Z}_1 \cup \mathbf{Z}_2$, then $eval(t) = \{ \mathbf{y} \in \mathbf{D}^p \mid s * eval(\{ \mathbf{Z}_1 : \forall \mathbf{Z}_2 \mathcal{R}(t_1, \dots, t_n)[\mathbf{Y}/\mathbf{y}, \mathbf{Z}_1, \mathbf{Z}_2] \}) \}$;
 - $eval$ is not defined in all other cases;

¹ The semantics of the various comparison operators $*$ on evaluated terms is the classical interpretation of comparisons between constants

- $\alpha[\mathbf{X}/\mathbf{x}] = \alpha_1[\mathbf{X}/\mathbf{x}], \dots, \alpha_p[\mathbf{X}/\mathbf{x}]$, $p \geq 0$, for which variables \mathbf{X} are replaced by constants \mathbf{x} , is evaluated (i) to true if $\forall i, 1 \leq i \leq p$, $\alpha_i[\mathbf{X}/\mathbf{x}]$ is evaluated to true, and (ii) to false otherwise;
- $\forall \mathbf{X} (\alpha[\mathbf{X}] \rightarrow \beta_{min} \leq \#(\{\mathbf{Y} : \gamma[\mathbf{X}]\}) \leq \beta_{max})$, say with $|\mathbf{X}| = p \geq 0$, is evaluated (i) to true if for each $\mathbf{x} \in \mathbf{H}_k^p$, either both $\alpha[\mathbf{X}/\mathbf{x}]$ and $\beta_{min} \leq \#(\{\mathbf{Y} : \gamma[\mathbf{X}/\mathbf{x}]\}) \leq \beta_{max}$ are evaluated to true or $\alpha[\mathbf{X}/\mathbf{x}]$ is evaluated to false, and (ii) to false otherwise — \mathbf{x} is any p -tuple on the Herbrand universe as it can be a set.

Observe that in the above definitions we have used untyped tuples to simplify notation - this simplification is not a restriction as the correct domain type is eventually enforced by the occurrence of tuple values in the appropriate domain in the relation \mathcal{R} . Also note that, on the basis of its semantics, $\{\mathbf{Z}_1 : \forall \mathbf{Z}_2 \mathcal{R}(t_1, \dots, t_n)\}$, is actually a shorthand for $\{\mathbf{Z}_1 : \bigwedge_{Z \in \mathbf{Z}_1} \mathcal{D}(Z) \wedge \forall \mathbf{Z}_2 \mathcal{R}(t_1, \dots, t_n)\}$.

We finally remark that the actual range of a variable X in a count constraint is restricted by the bound conditions we have introduced — we can then say that computation is in general safe. A potential range explosion could arise in a comparison predicate $X \subset t$ or $X \subseteq t$, but this risk is mitigated for the term t eventually unifies with a constant set of tuples and by the restriction on the maximum set arity \hat{k} .

A relation r satisfies a count constraint C (and we write $C \models r$) if the evaluation of C on r is equal to true. Accordingly, r satisfies a set of count constraints \mathbf{C} (and we write $\mathbf{C} \models r$) if for each $C \in \mathbf{C}$, $C \models r$.

As stated in the proposition below, checking a count constraint satisfaction may require exponential time.

Proposition 10 *Given a relation scheme $R(A_1, \dots, A_n)$ with $n > 0$, the domains D_1, \dots, D_n with possibly a fixed additional number of hierarchy domains, a maximal set arity \hat{k} , a count constraint C and a relation r on R , deciding $C \models r$ is in EXP.* \square

Proof.

Let C be a count constraint:

$$\forall \mathbf{X} (\alpha \rightarrow \beta_{min} \leq \#(\{\mathbf{Y} : \gamma\}) \leq \beta_{max}).$$

The size of the input is $d + c + g$, where d is the total size of the domains, c is the size of the constraint and g is the size of the relation r . Let $\mathbf{H}_{\hat{k}}$ be the Herbrand universe and let h be the size of $\mathbf{H}_{\hat{k}}$. Let $v \leq c$ be the number of variables in \mathbf{X} .

Consider the following procedure:

Count Constraint Check Procedure

- i set $check = \text{true}$;
- ii for each s in \mathbf{H}^v and while ($check$ is true)
 - 2.1 set $check_\alpha = \text{eval}(\alpha)$;
 - 2.2 if ($check_\alpha$ is true)
 - set $check = \text{eval}(\beta_{min} \leq \#(\{\mathbf{Y} : \gamma\}) \leq \beta_{max})$;
- iii return $check$;

Steps 2.1 can be done in time polynomial in $d \times c$, Step 2.2 in time polynomial in g . Steps 2.1 and 2.2 are repeated as many times as the number of elements in $\mathbf{H}_{\hat{k}}^v$, say p .

Suppose first that C is a tuple count constraint. Then $h \leq d$; so $p \leq d^v \leq d^c$. So the overall procedure runs in time polynomial in $d^c \times (d \times c + g)$ and, therefore, it is exponential in the size of the input.

Suppose now that C is a group count constraint. Then $h = \mathcal{O}(2^{d^{\hat{k}}})$; so $p = \mathcal{O}((2^{d^{\hat{k}}})^v) = \mathcal{O}(2^{v \times d^{\hat{k}}})d^{n \times v}$. As \hat{k} is a constant, p is exponential in the size of the input. Hence, as the overall procedure runs in time polynomial in $p \times (d \times c + g)$, the check is done in time exponential in the size of the input also in this case. \square

Particularly interesting are constraints for which the number of iterations p is polynomial, thus the structure of such constraints allows us to perform the iterations only over a polynomial subset of the domain. We say that count constraints with this property are *well structured*.

4.3.1 A Motivating Example

We refer to a classical example of point-of-sales transaction star schema. The attributes of U are: T (Transaction), I (Item), B (Brand), S (Store), A (Area) — their (finite) domains can be suitably defined. We are also given the following functional dependencies (FDs): $T \rightarrow S$, $S \rightarrow A$. It turns out that $\{T, I, B\}$ is the relation key. The domains of the attributes are denoted by \mathcal{D}_T , \mathcal{D}_I and so on. We next present a number of meaningful count constraints that clarify their usage:

(i): Enforcing FDs and relation key

We can use count constraints to enforce the FDs. For instance $T \rightarrow S$ can be expressed as follows:

$$\forall T (\mathcal{D}_T(T) \rightarrow 0 \leq \#(\{S : \forall I, B, A \mathcal{R}(T, I, B, S, A)\}) \leq 1)$$

The relation key $\{T, I, B\}$ can be enforced as:

$$\begin{aligned} \forall T, I, B (\mathcal{D}_T(T) \wedge \mathcal{D}_I(I) \wedge \mathcal{D}_B(B) \rightarrow \\ 0 \leq \#(\{S, A : \mathcal{R}(T, I, B, S, A)\}) \leq 1) \end{aligned}$$

(ii): Enforcing the overall number of tuples

There must be between 50000 and 100000 tuples in any instance of R :

$$\rightarrow 50000 \leq \#(\{T, I, B, S, A : \mathcal{R}(T, I, B, S, A)\}) \leq 100000$$

(iii): Enforcing the total number of transactions in an area

There must be between 1000 and 2000 transactions in every region, except in "Cal" (Calabria, not California!) for which the upper bound is increased to 9000:

$$\begin{aligned} \rightarrow 1000 \leq \#(\{T : \forall I, B, S \mathcal{R}(T, I, B, S, \text{"Cal"})\}) \leq 9000; \\ \forall A (\mathcal{D}_A(A) \wedge A \neq \text{"Cal"} \rightarrow \\ 1000 \leq \#(\{T : \forall I, B, S \mathcal{R}(T, I, B, S, A)\}) \leq 2000). \end{aligned}$$

If we wish to enforce the above transaction constraint in every store of an area, we can use the dimension hierarchy domain $\mathcal{D}_{S,A}$:

$$\begin{aligned}
& \forall S (\mathcal{D}_{S,A}(S, \text{"Cal"}) \rightarrow \\
& \quad 1000 \leq \#(\{T : \forall I, B \mathcal{R}(T, I, B, S, \text{"Cal"})\}) \leq 9000); \\
& \forall A, S (\mathcal{D}_{S,A}(S, A) \wedge A \neq \text{"Cal"} \rightarrow \\
& \quad 1000 \leq \#(\{T : \forall I, B \mathcal{R}(T, I, B, S, A)\}) \leq 2000).
\end{aligned}$$

(iv): 1-arity group count constraints

Both the set of items $i = \{[a], [b], [c]\}$ and $j = \{[b], [c], [d]\}$ must be present in at least 100 and in at most 200 transactions, whereas every other set s of items cannot appear in more than 15 transactions if s contains more than 10 elements or 20 otherwise, except for all subsets of i and j that have no limits (for space reasons we below write i and j to represent the two constant set terms):

$$\begin{aligned}
& \forall \hat{I} (\hat{I} = i \vee \hat{I} = j \rightarrow \\
& \quad 100 \leq \#(\{T : \hat{I} \subseteq \{I : \forall B, S, A \mathcal{R}(T, I, B, S, A)\}\}) \leq 200); \\
& \forall \hat{I} (\hat{I} \subseteq \{I : \mathcal{D}_I(I)\} \wedge \hat{I} \not\subseteq i \wedge \hat{I} \not\subseteq j \wedge \#(\hat{I}) \leq 10 \rightarrow \\
& \quad 0 \leq \#(\{T : \hat{I} \subseteq \{I : \forall B, S, A \mathcal{R}(T, I, B, S, A)\}\}) \leq 20); \\
& \forall \hat{I} (\hat{I} \subseteq \{I : \mathcal{D}_I(I)\} \wedge \hat{I} \not\subseteq i \wedge \hat{I} \not\subseteq j \wedge \#(\hat{I}) > 10 \rightarrow \\
& \quad 0 \leq \#(\{T : \hat{I} \subseteq \{I : \forall B, S, A \mathcal{R}(T, I, B, S, A)\}\}) \leq 15).
\end{aligned}$$

Note that the first of the above constraints has a disjunction in the left hand side: it is only a shorthand to represent two constraints having the same right hand side.

Observe that constraint define an instance of an IFM problem for which, in addition to fixing support constraints for a number of pre-defined itemsets (typically the frequent ones, in this case i and j), there are generic support constraints for all other itemsets (the infrequent ones).

If we now replace the operator \subseteq with $=$ in the count terms on the right hand sides of the above constraints, then the count bounds hold only when an itemset indicated by \hat{I} contain exactly all items of a transaction. So, in terms of the mentioned IFM problem, this revised formulation introduces duplicate constraints.

(iv): 2-arity group count constraints

There must be at least 100 and at most 200 transactions containing an item "sm" (smartphone) of the brand "nd" (ndrangtung), whereas the same set of pairs of item and brand are sold together in at most 10 transactions, except for the ones containing the pair ("sm", "nd") for which the limit is 50 (for space reasons we write t to represent the singleton constant set term $\{["sm", "nd"]\}$):

$$\begin{aligned}
& \rightarrow 100 \leq \#(\{T : t \subseteq \{I, B : \forall S, A \mathcal{R}(T, I, B, S, A)\}\}) \leq 200; \\
& \forall \hat{X} (\hat{X} \subseteq \{I, B : \mathcal{D}_I(I) \wedge \mathcal{D}_B(B)\} \wedge t \subset \hat{X} \rightarrow \\
& \quad 0 \leq \#(\{T : \hat{X} \subseteq \{I, B : \forall S, A \mathcal{R}(T, I, B, S, A)\}\}) \leq 50); \\
& \forall \hat{X} (\hat{X} \subseteq \{I, B : \mathcal{D}_I(I) \wedge \mathcal{D}_B(B)\} \wedge t \not\subset \hat{X} \rightarrow \\
& \quad 0 \leq \#(\{T : \hat{X} \subseteq \{I, B : \forall S, A \mathcal{R}(T, I, B, S, A)\}\}) \leq 10).
\end{aligned}$$

(Recall that \subset denotes strict subset relationship.) The above constraints define an instance of an IFM problem in which classical itemsets are replaced by sets of object pairs.

4.4 The Inverse OLAP Problem: Definition and Complexity

In this section we define a new inverse problem over a star schema that is relevant for OLAP applications.

Problem 4.1 (Inverse OLAP).

Given a relation scheme $R(A_1, \dots, A_n)$ with $n > 0$, the domains D_1, \dots, D_n with possibly a fixed additional number of hierarchy domains, a maximum set arity \hat{k} and a set of general count constraints C on R , the *Inverse OLAP problem* consists in deciding whether there exists a relation r on R such that $C \models r$. \square

Proposition 11 *The Inverse OLAP problem is in NEXP.*

Proof sketch.

Let p be the size of the input. A nondeterministic machine can guess a table r over R . The number k of tuples in r will be at most $d^1 \times \dots \times d^n$ tuples, that is exponential in p . Satisfaction of each general count constraints in C can be done in exponential time in p by Count Constraint Check Procedure. Hence the overall procedure is executed in exponential time by a non-deterministic machine. \square

In the next subsection, we shall prove that the problem is indeed NEXP-complete. We point out that, in our general setting, we are considering the so-called "combined complexity" [Var82]: both the number of attributes, the size of constraints and the domain sizes are not fixed and are part of the input. The "program complexity" version of the problem consists in considering domain sizes as constants. On the other hand, the "data complexity" version of the inverse OLAP problem fixes the number of attributes and the size of constraints and considers domain sizes as the only problem input.

In the next sub-section we present a simple case of the Inverse OLAP problem with binary domains and prove its NEXP-hardness. It will then turn out that the "program complexity" of Inverse OLAP is NEXP-complete and, therefore, the Inverse OLAP problem is NEXP-complete under the combined complexity as well. In addition, as the constraints we shall use are well structured, we also derive that NEXP-hardness holds if we restrict our attention to well-structured constraints only.

4.4.1 Binary Domain Inverse OLAP

In this section we assume that all attributes A_1, \dots, A_n in U have the same binary domain $\mathcal{D} = \{0, 1\}$. We denote a k -tuple of 0-1 values by a^k and the i -th element of a^k by $a^k[i]$. We also assume that, let $X = \{A_{i_1}, \dots, A_{i_m}\}$ be a subset of attributes in U and $\{A_{y_1}, \dots, A_{y_k}\}$ be the attributes in $U \setminus X$, count constraints have one of the following two formats:

- *fixed tuple count constraint*, briefly denoted by (X, a^k, l, h) , in the following form:

$$A_{i_1} = a^k[1] \wedge \dots \wedge A_{i_m} = a^k[m] \rightarrow \\ l \leq \#(\{A_{y_1}, \dots, A_{y_k} : \mathcal{R}(A_1, \dots, A_n)\}) \leq h$$

- *generic tuple count constraint*, briefly denoted by (X, l, h) , in the following form:

$$\forall A_{i_1}, \dots, A_{i_m} : \mathcal{D}(A_{i_1}) \wedge \dots \wedge \mathcal{D}(A_{i_m}) \rightarrow \\ l \leq \#(\{A_{y_1}, \dots, A_{y_k} : \mathcal{R}(A_1, \dots, A_n)\}) \leq h$$

Example 4.2. Consider the binary relation R on attributes A_1, A_2, A_3, A_4 , let $C_1 = (\{A_1, A_2\}, [1, 0], 1, 2)$ be a fixed tuple count constraint and $C_2 = (\{A_1, A_3\}, 1, 2)$ be generic tuple count constraint, then C_1 is represented in the following way:

$$A_1 = "1" \wedge A_2 = "0" \rightarrow \\ 1 \leq \#(\{A_3, A_4 : \mathcal{R}(A_1, A_2, A_3, A_4)\}) \leq 2$$

and C_2 is represented in the following way:

$$\forall A_1, A_3 : \mathcal{D}(A_1) \wedge \mathcal{D}(A_3) \rightarrow \\ 1 \leq \#(\{A_2, A_4 : \mathcal{R}(A_1, A_2, A_3, A_4)\}) \leq 2$$

□

It is easy to see that both fixed and generic tuple count constraints are well-structured since the count constraint check procedure can be done in polynomial time.

Problem 4.3 (Binary Domain Inverse OLAP).

Given a relation scheme $R(A_1, \dots, A_n)$ on binary domains, a set of fixed tuple count constraints C on R and a set of generic tuple count constraints C' on R , the *Binary Domain Inverse OLAP* problem consists in deciding whether there exists a relation r on R such that $(C, C') \models r$. □

Theorem 2 *The Binary Domain Inverse OLAP problem is NEXP-complete.*

Proof.

Membership to NEXP immediately derives from Proposition 11.

To prove NEXP-hardness we use a reduction of the Succinct Hamiltonian Cycle Problem, that has been proved to be NEXP-complete [PY86],[Pap94]. In this problem a graph G_{BC} , say with 2^n nodes, is represented in a succinct way. The nodes are coded with n bits and the edges are defined by a Boolean circuit B_C with $2n$ input and one output such that, the output value is 1 if and only if the pair of nodes described by the $2 * n$ bits of the circuit input is connected by an edge. The *Succinct Hamiltonian Cycle Problem (SHC)* is formulated as follows: given a Boolean circuit B_C with k gates, $2n$ inputs and one output, does the graph thus represented have an Hamiltonian cycle?

We next exhibit a reduction from SHC following the lines used by Kolaitis and Papadimitriou in [KP88] to prove that deciding whether a DATALOG program with negation has a fixpoint or not is NEXP-complete.

Given a graph G_{BC} , say with 2^n nodes, its succinct representation is done as follows. The nodes are coded with n bits and the edges are defined by a Boolean circuit with $2n$ input and one output such that, the output value is 1 if and only if the pair of nodes described by the $2 * n$ bits of the circuit input is connected by an edge. The boolean circuit encoding the graph contains k gates and is defined as a set of quadruples $B_C = \{G_u = (u, a_u, in_u^1, in_u^2) \mid u = 1, \dots, k\}$, where $k > 2n$ is the number of gates, u identifies the gate, $a_u \in \{OR, AND, NOT, IN\}$ is the kind of gate and in_u^1, in_u^2 are its input. The first $2n$ gates are of kind *IN*. According to the value of a_u , the meaning of in_u^1 and in_u^2 change:

- if $a_u \in \{OR, AND\}$ then in_u^1 and in_u^2 represent the two inputs of the gate, encoded by the indices of the two gates, whose outputs enter a_u — for instance, $(7, AND, 5, 9)$ denotes that the inputs of the *AND* gate 7 are the outputs of gates 5 and 9;
- if $a_u = NOT$ then $in_u^1 = in_u^2$ represents the unique input of the gate — for instance, $(9, NOT, 5, 5)$ denotes that the input of the *NOT* gate 9 is the output of gate 5;
- if $a_u = IN$ then $in_u^1 = in_u^2 = 0$ and u represents the $(i - 1)$ -th input bit of the circuit — for instance, $(3, IN, 0, 0)$ denotes that the input of the *IN* gate 3 is the second bit of the circuit (note that the input bits are numbered starting from 0 whereas the gates from 1).

Let $B_C = \{G_u = (u, a_u, in_{1,u}, in_{2,u}) \mid u = 1, \dots, k\}$ be any instance of SHC with $2n$ input gates. We build in polynomial time an instance of Binary Domain Inverse OLAP as follows. The relation schema R contains the following $5n + k$ attributes:

- $A_1, \dots, A_n, A_{n+1}, \dots, A_{2n}$: sub-tuples on them encode pairs of nodes that are entered into the circuit inputs $0, \dots, 2n - 1$ — we shall introduce constraints such that a sub-tuple t on these attributes indicates the presence of an edge from the node coded by $t.[A_1, \dots, A_n]$ to the node coded by $t.[A_{n+1}, \dots, A_{2n}]$; eventually the constraints will select sub-tuples encoding an Hamiltonian circuit (if any);
- $B_1, \dots, B_n, B_{n+1}, \dots, B_{2n}$: sub-tuples on them encode fictitious pairs of nodes — we shall introduce constraints to construct the cycle $\langle (0, 1), (1, 2), \dots, (2^n - 1, 2^n - 1), (2^n - 1, 0) \rangle$, that will serve to enforce that the sub-tuples on the A_i attributes (see the previous point) form a Hamiltonian circuit; the constraints will construct the sub-tuples starting from the node 0 by implementing a simple increment operator;
- C_1, \dots, C_k : sub-tuples on them encode the output values of the k gates;
- D_1, \dots, D_{n-1} : sub-tuples on them encode the possible $n - 1$ remainders for the increment operator (see the description for B_i attributes);
- E : the value on this attribute enables to divide a table on r into two horizontal fragments: we call them fragment 0 and fragment 1.

Let us now construct the set of fixed tuple count constraints C . They are divided into 3 groups :

Group (I): The constraints implement gate operations on the values represented in the C_i attributes, by enforcing that the wrong results cannot be present in a table - the constraints operate on the fragment 0 of a table, singled out by setting the value of column E to 0:

- i for each gate $G_u = (u, AND, in_u^1, in_u^2) \in B_C$ — the constraints enforce support 0 for the negated AND truth table (we represent the 4 constraints in parametric way):
 - $(\{C_u, C_{in_u^1}, C_{in_u^2}, E\}, \alpha, 0, 0)$, for each $\alpha = [0, 1, 1, 0], [1, 0, 0, 0], [1, 1, 0, 0], [1, 0, 1, 0]$
- ii for each gate $G_u = (u, OR, in_u^1, in_u^2) \in B_C$ — the constraints enforce support 0 for the negated OR truth table:
 - $(\{C_u, C_{in_u^1}, C_{in_u^2}, E\}, \alpha, 0, 0)$, for each $\alpha = [0, 1, 1, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0]$
- iii for each gate $G_u = (u, NOT, in_u^1, in_u^2) \in B_C$:
 - $(\{C_u, C_{in_u^1}, E\}, \alpha, 0, 0)$, for each $\alpha = [0, 0, 0], [1, 1, 0]$
- iv for each gate $G_u = (u, IN, in_u^1, in_u^2) \in B_C$ — the constraints enforce the support 0 for the negated value of the $(u - 1)$ -th bit, that is the input to G_u :
 - $(\{C_u, A_u, E\}, \alpha, 0, 0)$, for each $\alpha = [0, 1, 0], [1, 0, 0]$
- v this constraint enforces the fragment 0 to have 2^n "yes" outputs (stored into the C_k column), so that the fragment eventually stores 2^n edges of the graph in the A_i columns :
 - $(\{C_k, E\}, [1, 0], 2^n, 2^n)$

Group (II): This group too of constraints works on fragment 0 and enforces the construction of the fictitious cycle to be stored on the B_i attributes — given a sub-tuple on attributes B_1, \dots, B_n representing a node, say x , the constraint enforces that the sub-tuple on attributes B_{n+1}, \dots, B_{2n} of the same tuple represents the node $x+1$, by implementing an increment operator (we use the usual approach of imposing support 0 to the results that are not performed by the operator); we also observe that operator computes the node 0 as successor of the last node:

- i $(\{B_n, B_{2n}, E\}, \alpha, 0, 0)$ for each $\alpha = [1, 1, 0], [0, 0, 0]$ — increment of the low order bit of the node x
- ii for $i = 1, \dots, n - 1$ — increment of the other $n - 1$ bits of the node x using remainders stored on attributes D_i
 - $(\{B_i, D_i, B_{n+i}, E\}, \alpha, 0, 0)$ for each $\alpha = [0, 0, 1, 0], [1, 0, 0, 0], [0, 1, 0, 0], [1, 1, 1, 0]$
- iii $(\{B_n, D_{n-1}, E\}, \alpha, 0, 0)$ for each $\alpha = [1, 0, 0], [0, 1, 0]$ — storing the remainder of the low order bit
- iv for $i = 2, \dots, n - 1$ — storing the remainders of the other bits except the highest order bit, whose remainder can be neglected:
 - $(\{B_i, D_i, D_{i-1}, E\}, \alpha, 0, 0)$ for each $\alpha = [0, 0, 1, 0], [1, 0, 1, 0], [0, 1, 1, 0], [1, 1, 0, 0]$

Group (III): These constraints work on fragment 1 and enforce that the two edges on A_i and B_i attributes, respectively, be self loops, i.e., respectively (x, x) and (y, y) with x not necessarily different from y — these constraints will serve for constructing a bijection between A_i nodes and B_i nodes so that, because of the bijection and of the fictitious cycle on B_i , a isomorphism is realized for which the edges on A_i form a Hamiltonian cycle (more details on the issue will be given later in the proof, here we only anticipate that the node y is actually encoded using one's complement):

- i For $i = 1, \dots, n$:
 - $(\{A_i, A_{n+i}, E\}, \alpha, 0, 0)$ for each $\alpha = [1, 0, 1], [0, 1, 1]$ — enforcing (x, x) on A_i attributes
 - $(\{B_i, B_{n+i}, E\}, \alpha, 0, 0)$ for each $\alpha = [1, 0, 1], [0, 1, 1]$ — enforcing (y, y) on B_i attributes

Let us now construct the set of generic tuple count constraints C' . There are two groups of constraints, all working on both fragments 0 and 1: **Group (IV):** These constraints enforce that, for each of the two fragments 0 and 1, all nodes are stored into the A_i and B_i columns exactly once:

- i $(\{A_1, \dots, A_n, E\}, 1, 1)$ — first node in the pair stored on A_i attributes
- ii $(\{A_{n+1}, \dots, A_{2n}, E\}, 1, 1)$ — second node in the pair stored on A_i attributes
- iii $(\{B_1, \dots, B_n, E\}, 1, 1)$ — first node in the pair stored on B_i attributes
- iv $(\{B_{n+1}, \dots, B_{2n}, E\}, 1, 1)$ — second node in the pair stored on B_i attributes

Group (V): These constraints, together with the ones of Group (III), realize the bijection between A_i nodes and B_i , thus enforcing the construction of a Hamiltonian circuit — they work for the whole table and not separately for each of the two fragments as in the Group (IV):

- i For each B_j with $j = 1, \dots, n$
 - $(\{A_1, \dots, A_n, B_j\}, 1, 1)$ — a first node in a pair stored on A_i attributes (say x) is associated to exactly one of the first nodes in the pairs stored on B_i attributes, say y ; moreover, since x is present once in fragment 0 and once in fragment 1, the association inside fragment 1 is between x and the one's complement of y in order to respect the constraints;
- ii For each B_i with $i = n + 1, \dots, 2n$
 - $(\{A_{n+1}, \dots, A_{2n}, B_i\}, 1, 1)$ — association for the second node in a pair stored on A_i attributes; note that, because of constraints of Group (III), the association for the node is the same as the one enforced when the node appears as first node in a pair (see constraints 1 of this group).

As mentioned before, the constraints enforces that a table r on R has 2^{n+1} tuples and can be segmented by a selection on E into fragments 0 and 1, each with 2^n tuples: $F_0 = \sigma_{E=0}(r)$ and $F_1 = \sigma_{E=1}(r)$. Let us now further consider the following sub-fragments of r obtained by suitably projecting the fragments F_0 and F_1 :

$$\begin{aligned}
 \mathbf{A} &= \pi_{\mathbf{A}_1, \dots, \mathbf{A}_{2n}}(\mathbf{F}_0) & \mathbf{B} &= \pi_{\mathbf{A}_1, \dots, \mathbf{A}_{2n}}(\mathbf{F}_0) \\
 \mathbf{A}_1 &= \pi_{\mathbf{A}_1, \dots, \mathbf{A}_n}(\mathbf{A}) & \mathbf{A}_2 &= \pi_{\mathbf{A}_{n+1}, \dots, \mathbf{A}_{2n}}(\mathbf{A}) \\
 \mathbf{B}_1 &= \pi_{\mathbf{B}_1, \dots, \mathbf{B}_n}(\mathbf{B}) & \mathbf{B}_2 &= \pi_{\mathbf{B}_{n+1}, \dots, \mathbf{B}_{2n}}(\mathbf{B}) \\
 \mathbf{A}' &= \pi_{\mathbf{A}_1, \dots, \mathbf{A}_{2n}}(\mathbf{F}_1) & \mathbf{B}' &= \pi_{\mathbf{A}_1, \dots, \mathbf{A}_{2n}}(\mathbf{F}_1) \\
 \mathbf{A}'_1 &= \pi_{\mathbf{A}_1, \dots, \mathbf{A}_n}(\mathbf{A}') & \mathbf{A}'_2 &= \pi_{\mathbf{A}_{n+1}, \dots, \mathbf{A}_{2n}}(\mathbf{A}') \\
 \mathbf{B}'_1 &= \pi_{\mathbf{B}_1, \dots, \mathbf{B}_n}(\mathbf{B}') & \mathbf{B}'_2 &= \pi_{\mathbf{B}_{n+1}, \dots, \mathbf{B}_{2n}}(\mathbf{B}')
 \end{aligned}$$

The sub-fragments $\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{A}'_1, \mathbf{A}'_2, \mathbf{B}'_1$ and \mathbf{B}'_2 are depicted in Figure 1:

$A_1..$ $..A_n$	$A_{n+1}..$ $..A_{2n}$	$B_1..$ $..B_n$	$B_{n+1}..$ $..B_{2n}$	$L..$ $..C_k$	$D_1..$ $..D_{n-1}$	E
						0
\mathbf{A}_1	\mathbf{A}_2	\mathbf{B}_1	\mathbf{B}_2			0
						0
						1
\mathbf{A}'_1	\mathbf{A}'_2	\mathbf{B}'_1	\mathbf{B}'_2			1
						1

Fig. 4.1. Table r on R .

Each of sub-fragments $\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{A}'_1, \mathbf{A}'_2, \mathbf{B}'_1$ and \mathbf{B}'_2 has n columns and 2^n tuples and stores all the 2^n nodes of the graph.

We have the following claims — with a little abuse of notation, given a tuple t and a fragment X , t_X will denote the projection of t on the attributes of X :

Claim. 2 For each tuple t in \mathbf{A} , the sub-tuples $t_{\mathbf{A}_1}$ and $t_{\mathbf{A}_2}$ represent two vertices u, v of the succinct graph such that there exists an edge between u and v in the graph.

Proof of the claim.

The count constraints of group (I) implements the Boolean Circuit and the constraint 4 of the group imposes that the circuit must have output value 1, thus there is an edge between u and v . □

Claim. 3 For each tuple t in \mathbf{B} , $\text{int}(t_{\mathbf{B}_2}) = \text{int}(t_{\mathbf{B}_1} + 1) \bmod 2^n$ where $\text{int}(t_{\mathbf{B}_1})$ and $\text{int}(t_{\mathbf{B}_2})$ are the non negative integer values represented by the binary tuples $t_{\mathbf{B}_1}$ and $t_{\mathbf{B}_2}$, respectively.

Proof of the claim.

The count constraints of group (II) implement a binary incremental circuit that take in input a tuple $t_{\mathbf{B}_1}$ and returns the tuple $t_{\mathbf{B}_2}$ such that $\text{int}(t_{\mathbf{B}_2}) = (\text{int}(t_{\mathbf{B}_1}) + 1) \bmod 2^n$. □

Claim. 4 Let $t \in F_0$ and $t' \in F_1$. Then:

1. if $t_{A_1} = t'_{A'_1}$ then $t'_{B'_1} = \text{neg}(t_{B_1})$ where $\text{neg}(t_{B_1})$ is the one's complement of tuple t_{B_1} ,
2. if $t_{A_2} = t'_{A'_2}$ then $t'_{B'_2} = \text{neg}(t_{B_2})$.

Proof of the claim.

The proof is a direct consequence of the generic tuple count constraints of group (V).
□

Claim. 5 For each tuple $t \in F_1$, $t_{A'_1} = t_{A'_2}$ and $t_{B'_1} = t_{B'_2}$.

Proof of the claim.

The proof is a direct consequence of the count constraints of group (III). □

By Claim 2, the tuples in **A** represent 2^n distinct edges of the graph G_{BC} such that each node has exactly two incident edges. However, this condition is not sufficient to establish the existence of Hamiltonian Cycle because the situation shown in Figure 4.2 may arise: each node of the graph in figure are two adjacent edges and the edges are 2^n but the graph is not a Hamiltonian cycle.

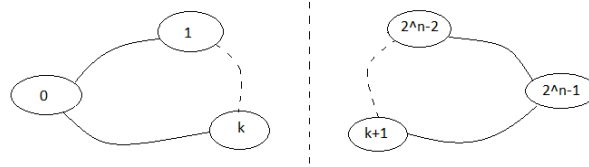


Fig. 4.2. It is not hamiltonian cycle ($0 < k < k + 1 < 2^n - 1$).

To actually enforce that the edges in **A** form a Hamiltonian cycle, we have constructed a fictitious Hamiltonian cycle in **B** and defined a bijection between nodes in **A** and in **B** so to have an isomorphism between the two cycles. By Claim 3, **B** represents a lexicographic cycle from the node 0 to $2^n - 1$ and back to 0 — we stress that this cycle is not in general present in the graph.

By Claims 4 and 5, there must be an isomorphism between the edges in **A** and **B** and this implies that the graph must have a Hamiltonian circuit. We can then conclude that the succinct graph G_{BC} has a Hamiltonian circuit if and only if there exists a relation r on R that satisfies the count constraints in C and C' . This concludes the proof. □

From this theorem we immediately derive the following important corollary.

Corollary 2 *The Inverse OLAP problem is NEXP complete under both the scheme complexity and the combined complexity.* □

4.4.2 Binary Attribute Inverse OLAP

Let us now assume that U contains only two attributes, say $U = \{A_T, A_I\}$ with domains T and $\mathcal{I} = \{o_1, \dots, o_n\}$. Let Y denote any given subset of \mathcal{I} . We also assume that count constraints have one of the following two formats:

- *k-support constraint*, briefly denoted by $\sigma(Y, k, l, h)$, in the following form:

$$\begin{aligned} \forall I' : I' \subseteq Y \wedge \#(I') = k \rightarrow \\ l \leq \#\{T : I' \subseteq \{I : \mathcal{R}(T, I)\}\} \leq h \end{aligned}$$

- *k-duplicate constraint*, briefly denoted by $\delta(Y, k, l, h)$, in the following form:

$$\begin{aligned} \forall I' : I' \subseteq Y \wedge \#(I') = k \rightarrow \\ l \leq \#\{T : I' = \{I : \mathcal{R}(T, I)\}\} \leq h \end{aligned}$$

Problem 4.4 (Binary Attribute Inverse OLAP).

Given a relation scheme $R = \{A_T, A_I\}$ on domains T and \mathcal{I} , a set of support constraints Γ_σ on R and a set of duplicate constraints Γ_δ on R , the *Binary Attribute Inverse OLAP* problem consists in deciding whether there exists a relation r on R such that $(\Gamma_\sigma, \Gamma_\delta) \models r$. \square

Theorem 3 *The Binary Attribute Inverse OLAP is NEXP-complete.*

Proof.

Membership to NEXP is obvious. Let us now prove NEXP-hardness. To this end, we perform a reduction from the Binary Domain Inverse OLAP problem.

Consider any instance of the Binary Domain Inverse OLAP problem. We are given a relation scheme $R(A_1, \dots, A_n)$ on binary domains, a set C of fixed tuple count constraints and a set C' of generic tuple count constraints. We construct an instance of the Binary Attribute Inverse OLAP Problem as follows.

For each attribute A_i , $1 \leq i \leq n$, we introduce two items in \mathcal{I} : A_i^0 and A_i^1 , corresponding to the two values that can be taken by the attribute. The set of items \mathcal{I} then consists of $2 \times n$ items and any tuple t on R can be represented by an itemset I as follows: for each A_i , $1 \leq i \leq n$, I contains A_i^0 if $t.A_i = 0$ or A_i^1 otherwise. For example, given a relation $R(A, B, C)$, we have $\mathcal{I} = \{A^0, B^0, C^0, A^1, B^1, C^1\}$ and the tuple $[0, 1, 1]$ on R is represented by the itemset $\{A^0, B^1, C^1\}$. Next we enforce that all transactions represent tuples on R .

First of all, we have to enforce that no transaction contains two items corresponding to the same attribute. To this end, we include in Γ_σ the support constraint $\sigma(\{A_1^0, A_1^1\}, 2, 0, 0), \dots, \sigma(\{A_n^0, A_n^1\}, 2, 0, 0)$. Note that the support constraint $\sigma(Y, |Y|, l, h)$ means that in \mathcal{R} the number of transaction that transaction that contains L are between l and h .

Next we require that each transaction to have exactly n items by setting $\Gamma_\delta =: \{(\mathcal{I}, l, 0, 0) \mid 1 \leq l \leq 2 \times n, l \neq n\} \cup \{(\mathcal{I}, l, 0, 1) \mid l = n\}$. We also enforce the duplicate constraint $(0, 0)$ for all itemsets in S .

Now we express fixed tuple count constraints in terms of support constraints. Let (X, a^k, l, h) be a fixed count constraint in C , where $L = [A_{i_1}, \dots, A_{i_k}]$. We define I as $\{A_{i_j}^p \mid 1 \leq j \leq k \text{ and } p = 0 \text{ if } a^k[j] = 0 \text{ or } p = 1 \text{ otherwise}\}$ and insert the support constraint (I, k, l, h) into Γ_σ .

Let us now implement any generic tuple count constraint (X, l, h) in C' , where $L = [A_{i_1}, \dots, A_{i_k}]$. We define I as $\{A_{i_j}^0 \mid 1 \leq j \leq k\} \cup \{A_{i_j}^1 \mid 1 \leq j \leq k\}$ and include the constraint (I, k, l, h) into Γ_σ . The presence of specific constraints $(\{A_1^0, A_i^1\}, 0, 0)$ in Γ_σ overrules the generic constraints for all itemsets that do not correspond to tuples.

It is easy to see that the above procedure for constructing an instance of the Binary Attribute Inverse OLAP problem starting from an instance of the Binary Domain Inverse OLAP problem can be performed in time polynomial in the size of the latter instance. It is also easy to see that the instance of the Binary Attribute Inverse OLAP problem is a "yes" answer if and only if so is the instance of the Binary Domain Inverse OLAP problem. \square

4.4.3 Data Complexity of Inverse OLAP

In this section we will define a special case of Inverse OLAP problem, called Fixed Hierarchy Inverse OLAP, whose input instance is formed by a fixed number of hierarchy domains. The sizes of the schema relation and of the count constraints are fixed. This problem is useful in order to establish the Data Complexity of Inverse OLAP.

Problem 4.5 (Fixed Hierarchy Inverse OLAP).

Let R be a relation scheme of form

$$R(C_\sigma, Y_\sigma, K_\sigma, L_\sigma, H_\sigma, C_\delta, Y_\delta, K_\delta, L_\delta, H_\delta, T, I)$$

$\mathcal{D}_{C_\sigma, Y_\sigma}, \mathcal{D}_{C_\sigma, K_\sigma}, \mathcal{D}_{C_\sigma, H_\sigma}, \mathcal{D}_{C_\sigma, L_\sigma}, \mathcal{D}_{C_\delta, Y_\delta}, \mathcal{D}_{C_\delta, K_\delta}, \mathcal{D}_{C_\delta, H_\delta}, \mathcal{D}_{C_\delta, L_\delta}$ be the hierarchy domains, and C be a set of two count constraints of constant size, the *Binary Attribute Inverse OLAP* problem consists in deciding whether there exists a relation r on R such that $(C \models r)$. \square

Theorem 4 *The Fixed Hierarchy Inverse OLAP problem is NEXP-complete.*

Proof.

Membership to NEXP is obvious. Let us now prove NEXP-hardness. To this end, we perform a reduction from the Binary Attribute Inverse OLAP problem.

Consider any instance of the Binary Attribute Inverse OLAP problem. We are given a relation schema $R = \{A_T, A_I\}$ on domains T and \mathcal{I} , a set of k -support constraints $\Gamma_\sigma = \{\sigma_1, \dots, \sigma_p\}$ and a set of k -duplicate constraints $\Gamma_\delta = \{\delta_1, \dots, \delta_o\}$. We construct an instance of the Fixed Hierarchy Inverse OLAP Problem as follows.

For each support constraint $\sigma_i = \sigma(Y, k, l, h)$, with $i = 1, \dots, p$, we adopt the following procedure:

- for each domain element $a \in Y$ insert in input $\mathcal{D}_{C_\sigma, Y_\sigma}("i", "a")$,
- and insert $\mathcal{D}_{C_\sigma, K_\sigma}("i", "k")$, $\mathcal{D}_{C_\sigma, L_\sigma}("i", "l")$, $\mathcal{D}_{C_\sigma, H_\sigma}("i", "h")$.

As shown above, we use the hierarchy domains to take in input the k -support constrains. In the same way, it is possible to transform the k -duplicate constrains in elements of hierarchy domains $(\mathcal{D}_{C_\delta, Y_\delta}, \mathcal{D}_{C_\delta, K_\delta}, \mathcal{D}_{C_\delta, H_\delta}, \mathcal{D}_{C_\delta, L_\delta})$.

To this end, we impose only two constraints that generalize the k -support constraint and k -duplicate constraint taking the parameters of each k -support and k -duplicate constraint directly by the hierarchy domains. The fixed count constraints considered, respectively for k -support and k -duplicate constraints, are:

$$\begin{aligned} \forall C, I' : I' \subseteq \{I : \mathcal{D}_{C_\sigma, Y_\sigma}(C, I)\} \wedge \mathcal{D}_{C_\sigma, H_\sigma}(C, H) \wedge \\ \mathcal{D}_{C_\sigma, L_\sigma}(C, L) \wedge \mathcal{D}_{C_\sigma, K_\sigma}(C, K) \wedge \#(I') = K \rightarrow \\ L \leq \#(\{T : I' \subseteq \{I : \forall C_\sigma, \dots, H_\delta \mathcal{R}(C_\sigma, \dots, H_\delta, T, I)\}\}) \leq H \end{aligned}$$

$$\begin{aligned} \forall C, I' : I' \subseteq \{I : \mathcal{D}_{C_\delta, Y_\delta}(C, I)\} \wedge \mathcal{D}_{C_\delta, H_\delta}(C, H) \wedge \\ \mathcal{D}_{C_\delta, L_\delta}(C, L) \wedge \mathcal{D}_{C_\delta, K_\delta}(C, K) \wedge \#(I') = K \rightarrow \\ L \leq \#(\{T : I' = \{I : \forall C_\delta, \dots, H_\delta \mathcal{R}(C_\delta, \dots, H_\delta, T, I)\}\}) \leq H \end{aligned}$$

Consider now the first constraint: it imposes that for each constraint C a for each I' subset of Y

$(I' \subseteq \{I : \mathcal{D}_{C_\sigma, Y_\sigma}(C, I)\})$ of cardinality K ($\#(I') = K \wedge \mathcal{D}_{C_\sigma, K_\sigma}(C, K)$) must be contained in number of transactions between L ($\mathcal{D}_{C_\sigma, L_\sigma}(C, L)$) and H ($\mathcal{D}_{C_\sigma, H_\sigma}(C, H)$).

Note that the definitions of this count constraints are independent by $\Gamma_\delta, \Gamma_\sigma$.

It is easy to see that the above procedure for constructing an instance of the Fixed Hierarchy Inverse OLAP problem starting from an instance of the Binary Domain Inverse OLAP problem can be performed in time polynomial in the size of the latter instance. It is also easy to see that the instance of the Fixed Hierarchy Inverse OLAP problem is a "yes" answer if and only if so is the instance of the Binary Domain Inverse OLAP problem. \square

Since the schema relation and the count constraints are fixed in Fixed Hierarchy Inverse OLAP, then the following corollary holds.

Corollary 3 *The Inverse OLAP problem is NEXP- complete under data complexity.*

4.5 A Step towards Aggregate Data Exchange

In this sections we present some meaningful example of how count constraints can be exploited for aggregate data exchange. We first recall the classical data exchange setting $(S, T, \Sigma_{st}, \Sigma_t)$, where S is the source relational scheme, T is the target relational database scheme, Σ_T is a logical formula on T and Σ_{st} are source-to-target dependencies of the form $\forall \mathbf{X}(\phi_S(\mathbf{X}) \rightarrow \chi_T(\mathbf{X}))$, where $\phi_S(\mathbf{X})$ and $\chi_T(\mathbf{X})$ are

formula on S and T , respectively. More detailed descriptions of this setting and of related properties and results can be found in [FKP05, GLLR07, GN08, FKPT09, APR11].

Aggregate data exchange for preserving privacy

The target relational database scheme consists of a unique relation scheme, that is the one used in Section 4.3.1: $\mathcal{R}(T, I, B, S, A)$. Recall that the meaning of the attributes is: T (Transaction), I (Item), B (Brand), S (Store), A (Area), and that the following FDs hold: $T \rightarrow S, S \rightarrow A$.

The source relational database scheme consists of three relation schemes: $\mathcal{TR}(T, I, B)$, $\mathcal{ST}(T, S)$ and $\mathcal{AR}(S, A)$. Observe that this scheme is the normalized version of the target scheme.

We want that the target relation be the natural join of the source relation but, for privacy reasons, the associations between transactions and pairs of item and brand must be perturbed: the transactions IDs of the same store are permuted. For instance, if the store s has n transactions t_1, \dots, t_n , the block of item-brand pairs of a transaction t_i are moved to a transaction t_j , then the block of t_j is moved to another transaction and so on.

Let us first use the classical setting to implement two natural joins of the source relations instead of only one so that we can later perform a permutation of transactions inside every store:

$$\begin{aligned} \forall T, I, B, S, A (\mathcal{TR}(T, I, B) \wedge \mathcal{ST}(T, S) \wedge \mathcal{AR}(S, A) \rightarrow \\ \exists T' \mathcal{R}(T', I, B, S, A)); \\ \forall T, S, (\mathcal{ST}(T, S) \rightarrow \exists I, B, A \mathcal{R}(T, I, B, S, A)); \end{aligned}$$

We now use a count constraint to enforce that the total number of tuples in \mathcal{TR} is equal to the total number of tuples in \mathcal{R} so that the target relation cannot store additional tuples:

$$\begin{aligned} \forall C (C = \#(\{T, I, B : \mathcal{TR}(T, I, B)\}) \rightarrow \\ C = \#(\{T, I, B, S, A : \mathcal{R}(T, I, B, S, A)\})); \end{aligned}$$

So we have lost the correspondence between transactions in the source and in the target scheme but with the following constraint imposes that the original structure of transactions is preserved modulo permutation of transactions IDs:

$$\begin{aligned} \forall S, T, \hat{X} (\mathcal{ST}(T, S) \wedge \hat{X} = \{I, B : \mathcal{TR}(T, I, B)\} \rightarrow \\ \exists T' (\hat{X} = \{I, B : \forall A \mathcal{R}(T', I, B, S, A)\})). \end{aligned}$$

The above constraint can be equivalently rewritten using a count predicate instead of the existential quantifier:

$$\forall S, T, \hat{X} (\mathcal{ST}(T, S) \wedge \hat{X} = \{I, B : \mathcal{TR}(T, I, B)\} \rightarrow \\ 1 \leq \#\{T' : \hat{X} = \{I, B : \forall A \mathcal{R}(T', I, B, S, A)\} \}).$$

Thus the count predicates is able to emulate an existential quantifier.

Data exchange to an OLAP scheme

We now assume that the relation $\mathcal{R}(T, I, B, S, A)$ represents the source scheme. The target scheme is an OLAP scheme $\mathcal{SN}(S, I, B, N)$ that, for every store, represents in N the total number of item-brand pairs that are in all transactions of that store.

We aggregate data in the target relation using a count predicate in the following constraint:

$$\forall I, B, S, A, N (N = \#\{T : \mathcal{R}(T, I, B, S, A) \rightarrow \\ \mathcal{SN}(S, I, B, N) \});$$

A final count constraint imposes that the target relation cannot store additional tuples:

$$\forall C (C = \#\{S, I, B : \forall T, A \mathcal{R}(T, I, B, S, A) \rightarrow \\ C = \#\{S, I, B : \forall N \mathcal{SN}(S, I, B, N)\} \});$$

Datalog with frequency support goals

We introduce a simple extension of Datalog, called Datalog^{FS}, that enables us to query and reason about the number of distinct occurrences satisfying given goals, or conjunction of goals, in rules. This simple extension preserves all the desirable semantic and computational properties of logic-based languages, while significantly extending their application range to support page-rank and social-network queries.

5.1 Introduction

Due to the emergence of many important application areas we are now experiencing a major resurgence of interest in Datalog [Hel10, dMMAG, HGL11]. A first such research area focuses on the use of logic-based declarative specifications and design of Internet protocols and services [LCG⁺09]; recent extensions of this work seek to generalize this approach by developing Datalog-based foundations for parallel and distributed programming languages [Hel10]. On the Semantic Web front, a novelty of great interest is represented by the introduction of Linear Datalog for expressing and supporting efficiently subsets of Description Logic for reasoning in ontological queries [GOP11]. Yet another very important development is represented by the use of Datalog in program analysis [HGL11]. Furthermore, other lines of work exploring the execution of Datalog queries in new computational environments have studied the optimization of recursive queries in the MapReduce framework [ABC⁺11], and the declarative and operational semantics of continuous Datalog queries in Data Stream Management Systems [Zan11].

This torrent of new applications underscores the need to tackle and solve crucial Datalog problems that were recognized more than twenty years ago but still remain unsolved and restrict the range of practical effectiveness of this elegant declarative-programming paradigm. For database applications in particular, the most vexing of these problems is represented by the constraints placed upon aggregates in recursive Datalog programs. Indeed, with the introduction of OLAP functions and massive analytics for decision support and web mining, the usage of aggregates in modern information systems and web applications has grown by leaps and bounds—making

limitations upon aggregates increasingly undesirable. Therefore, Datalog extensions that can improve its ability to deal with aggregates would represent a big step forward. Technically, however, the problem is very challenging since the requirement of monotonicity is deep-rooted in the fixpoint semantics of Datalog, and the many solution approaches attempted by researchers in the past were not general and practical enough to gain wide acceptance.

In this chapter, we introduce Datalog^{FS} a generalization of Datalog that is very effective at expressing a broad range of new applications requiring count-based aggregates, such as Apriori, Bill of Materials, social networks, and Markov Chains. Concepts such as stratification and magic-sets, and in general all the techniques and methods that provide the enabling technology for traditional Datalog, remain valid and effective for Datalog^{FS}, paving the way for a faster deployment and spreading of this powerful extension.

5.2 Related Work

Logic-based query languages were quintessential in the introduction of the relational data model by E.F. Codd in the 70s; through major efforts in scalable implementation and query optimization this led to the development of relational DBMS in the 80s. Over time, the excitement generated by the extraordinary success of relational DBMS, on both research and business fronts, gave way to the realization that query languages more expressive than relational algebra and SQL-2 were needed; this motivated the work on Datalog and related languages that support rules and recursive queries. Recursive Datalog programs come with a simple and elegant least-fixpoint semantics, which is also equivalent to both the logic-based model-theoretic and proof-theoretic semantics of the program clauses [Llo87, ZCF⁺]. Furthermore, the least-fixpoint semantics can be efficiently supported by the iterated execution of the rules enhanced by:

- (i) The differential fixpoint (a.k.a. seminaive fixpoint) method [AHV95, Ram98, ZCF⁺, UW97] that avoids redundancy in the bottom-up computation (i.e., from the database to the goal) of the rules, and
- (ii) Top-down methods, such as the Magic-Sets methods and the specialization of left/right recursive rules, that propagate constants and constraints from query goals downwards to restrict and expedite the bottom-up computation [AHV95, Ram98, ZCF⁺, UW97].

However, these great properties only hold when the rules define monotonic transformations (w.r.t. set-containment), and negation or aggregates in rules compromise their monotonicity along with the nice properties above. Much research work has therefore sought ways to generalize (i) the semantics of Datalog and (ii) its efficient implementation when negation and aggregates are used in the rules. This work has produced the notion of *stratification* that satisfy both properties, and is also quite simple for users to master [AHV95, Ram98, ZCF⁺, UW97]. Unfortunately, stratification (into a finite number of strata) is too restrictive and does not allow the expression of

popular optimization and data-mining algorithms, which typically require the computation of extrema and counts in recursion. The importance of optimization and data-mining applications have therefore motivated much research work seeking to solve these problems. In general, said proposals follow three main approaches. The first consists in supporting infinite levels of stratifications using Datalog_{1S} programs [ZAO93, LLM98, ZCF⁺]; a second approach instead attempts to preserve the fix-point computation via continuous aggregates and non-deterministic *choice* constructs [GGZ91, GPSZ91, GZ01, AOT⁺03]. Finally, the third approach seeks to achieve monotonicity by using partial orders that are more general than set-containment [MPR90, RS97, Gel93]. These works developed very ingenious solutions that, while being technically very sound, did not gain widespread popularity in the field for a number of reasons. One reason could be that this research was indeed ‘before its time’; but on a more serious note, we observe that the approaches mentioned above require constructs and semantics that were quite complex for users to master and for the system to support (e.g., it was quite difficult for the compiler to decide if the program at hand was monotonic or not [Gel93]). On the other hand, Datalog^{FS} is timely, and manages to be friendly to both users and compilers.

This chapter is organized as follows. In the next section, we introduce Datalog^{FS} via some simple examples, and in Section 4, we define its formal semantics. In Section 5, we introduce constructs that support the assertion of facts and the derivation of predicates having multiple occurrences. In Section 6, we outline a simple plan for the efficient implementation of Datalog^{FS} and show how differential fixpoint and magic-set techniques extend naturally to Datalog^{FS}. In Section 7, we introduce the notion of scaling which allow us to reason with frequencies that are decimal numbers rather than integers. Then, in Section 8, we review some important new applications, such as social networks and Markov chains, that follow from these extensions.

5.3 Datalog^{FS} by Examples

For example, our database could contain facts as follows:

```

person(adam).   person(marc).   person(jerry).
person(tom).
son(marc, tom). son(marc, jerry). son(tom, eddy).
son(tom, adam). son(tom, john).

```

Then the following rule defines fathers that have at least two sons:

$$\text{twosons}(X) \leftarrow \text{person}(X), \text{son}(X, Y1), \text{son}(X, Y2), Y1 \neq Y2.$$

Datalog^{FS} allows the following equivalent expression for our twosons rule:

$$\text{twosons}(X) \leftarrow \text{person}(X), 2: [\text{son}(X, Y)].$$

The goal, $I: [\text{B-expression}]$, will be called a frequency support goal (an *FS Goal* for short), and $I:$ where I is a positive integer will be called its *FS-test clause*.

The bracketed expression `B-expression` can either consists of a single goal or a conjunction of goals.

The convenience of FS-goals becomes clear if we want to find people who have a large number of sons. For instance:

$$\text{sixsons}(X) \leftarrow \text{person}(X), 6: [\text{son}(X, Y)].$$

will retrieve all persons who have at least six sons (i.e., 6 is the frequency support required for the predicates or conjunction of predicate withing brackets).

Naturally, an equivalent rule can be expressed using the \neq operator. Indeed we can start as follows:

$$\text{sixsons}(X) \leftarrow \text{person}(X), \text{son}(X, Y_1), 5: [\text{son}(X, Y_2), Y_2 \neq Y_1].$$

and then proceed inductively, and obtain a rule containing six goals $\text{son}(X, Y_j)$, where $j = 1, \dots, 6$ and 6×5 goals specifying, that every Y must be different from (i.e., \neq) every other Y . Of course, if rather than paternity between people we are interested in links between web pages, which could easily be in number of thousands, it becomes clear that the approach based on \neq becomes totally impractical, and without FS-goals we would need a COUNT aggregate to deal with these applications. However, aggregates bring in the curse of non-monotonicity making the use of recursion very difficult if not impossible. In Datalog^{FS} our rules can instead be expressed as standard Horn rules (although long and impractical ones). Thus Datalog^{FS} preserves the standard monotonicity-based semantics of negation-free Datalog; moreover, as we shall see later, the well-known implementation techniques used for Datalog can also be extended to Datalog^{FS}.

The next examples clarify the meaning and the scope of variables in Datalog^{FS}. We will use the predicate `friend(X, Y)` denoting that the person with name X views the person with name Y as his/her friend (no assumption of symmetry is made):

Example 5.1. Pairs of friends (F1, F2) where F1 has at least three friends and, and F2 also has three friends:

$$\text{popularpair}(X, Y) \leftarrow \text{friend}(X, Y), \\ 3: [\text{friend}(X, V_1)], 3: [\text{friend}(Y, V_2)].$$

The next one requires that F1 and F2 have at least three friends in common (unlike the previous query where F1 and F2 are not required to have friends in common):

Example 5.2. The following rule returns the pairs of friends (F1, F2) who have at least three friends in common

$$\text{sharethree}(X, Y) \leftarrow \text{friends}(X, Y), \\ 3: [\text{friend}(X, V), \text{friend}(Y, V)].$$

There are two kinds of variables in rules with FS-goals. The first are those, such as X and Y in the rule of Example 5.2, that appear in the head of the rule or in some

goal not contained in a b-expression. These will be called *global* variables in the rule. Global variables are basically the universally qualified variables of the standard Horn Clauses, and have the whole rule as their scope. Thus the variables X and Y in Example 5.1 are also global for that rule.

The remaining variables are those that only appear in b-expressions and their scope is *local* to the b-expression where they appear. For instance V1 and V2 in Example 5.1, and V in Example 5.2 are local variables. These variables can be naturally viewed as existential variables with the following minimum frequency support constraint: *There exist at least K occurrences of b-expression*. For instance, Example 5.2 states that there exist at least 3 distinct V occurrences each denoting a person who is viewed as friend by both X and Y.

It is also important to remember that the scope of such existential variables is local to the b-expression where they appear: thus, in Example 5.1 replacing both V1 and V2 with a V would not change the meaning of our rule.

For a more interesting example, let us express the often used guideline that an assistant professor to be advanced to associate professor should have an H-index [Hir10] of 13 or higher. This can be expressed with the following rules:

Example 5.3. Our candidate must have authored at least 13 chapters each of which has been referenced at least 13 times. The database table `author(Author, PaperNo)` lists chapters (co-)authored by a person, while `refer(PnoFrom, PnoTo)` denotes that paper PnoFrom contains a reference to paper PnoTo.

$$\begin{aligned} \text{atleast13(Pno)} &\leftarrow 13: [\text{refer(PnFrom, Pno)}]. \\ \text{hindex13(Author)} &\leftarrow 13: [\text{author(Author, Pno)}, \\ &\quad \text{atleast13(Pno)}]. \end{aligned}$$

These simple examples could also be easily expressed using the count aggregate. However count and other aggregates are non-monotonic and thus cannot be used in recursive rules. Indeed, the meaning and efficient implementation, of Datalog programs with recursive rules, are based on their least fixpoint semantics¹, which is only guaranteed to exist when the program rules define monotonic mappings. A solution to this problem proposed by Ross and Sagiv [RS97] to exploit partial orderings between sets that are different than the standard set-containment ordering used to define the semantics of Datalog and other logic programs. While several interesting programs with aggregates are in fact monotonic under this revised notion, many others are not and it would be quite difficult for a compiler to decide which is which [Gel93]. As we shall see next, the standard notion of monotonicity w.r.t set-containment ordering is instead used for Datalog^{FS} programs, whereby the basic properties and techniques of deductive databases (e.g., stratification and differential fixpoint) remain valid.

¹ Naturally, by “least fixpoint” of a program, we mean “least fixpoint of its immediate consequence operator” [Llo87].

5.4 Semantics of Datalog^{FS}

In this section, we define the semantics of Datalog^{FS} programs by their rewriting into equivalent Datalog programs. The rewritings of `twosons` and `sixsons` rules in Section 5.3 are correct but require the integer in the FS-test clause to be a constant. We next specify a re-writing that does not depend on this assumption.

5.4.1 Rewriting of Datalog^{FS} into Datalog

Each frequency support goal will be re-written separately. Thus, let the j^{th} such a goal be

$$K_j : [\text{expr}_j(X_j, Y_j)]$$

where X_j and Y_j , respectively denote the global variables and the local ones. For instance, for the FS-goal in the first rule of Example 5.3 then, the 13 is its FS-test clause, `PnFrom`, and `Pno` are respectively the global variables and the local variables. Then, our rewriting replaces $K_j : [\text{expr}_j(X_j, Y_j)]$ in the rule by `conj(Kj, Xj, -)` where the underscore denotes the anonymous variable and `conj` is defined as follows:

$$\begin{aligned} \text{conj}(1, X_j, [Y_j]) &\leftarrow \text{expr}_j(X_j, Y_j). \\ \text{conj}(N1, X, [Y_j|T]) &\leftarrow \text{expr}_j(X_j, Y_j), \text{conj}(N, X_j, T), \\ &\quad \text{notin}(Y_j, T), N1 = N + 1. \\ \text{notin}(Z, []) &. \\ \text{notin}(Z, [V|T]) &\leftarrow Z \neq V, \text{notin}(Z, T). \end{aligned}$$

For instance, the second rule in Example 5.3 becomes:

$$\text{hindex13}(\text{Author}) \leftarrow \text{con2}(13, \text{Author}, -).$$

Where `con2` is defined as follows:

$$\begin{aligned} \text{con2}(1, \text{Author}, [\text{Pno}]) &\leftarrow \text{author}(\text{Author}, \text{Pno}), \\ &\quad \text{atleast13}(\text{Pno}). \\ \text{con2}(N1, \text{Author}, [\text{Pno}|T]) &\leftarrow \text{author}(\text{Author}, \text{Pno}), \\ &\quad \text{atleast13}(\text{Pno}), \\ &\quad \text{con2}(N, \text{Author}, T), \\ &\quad \text{notin}(\text{Pno}, T), \\ &\quad N1 = N + 1. \end{aligned}$$

The rules defining `notin` are instead generic and can be shared by the different `conj` definition. In this example, we only have one local variable and one global variable. However, generalizing to the case where we have an arbitrary number of global variables and local variables is straightforward, since we can arranged then into a list of local variables and global variables. The definition of `notin` use goals such as $Z \neq V$ denoting inequality between single variables. In general, we might have several local variables which can be organized in lists (or other complex object).

Then the inequality condition between two lists can simply express the fact that not all the corresponding elements in the list are equal.

This rewriting of Datalog^{FS} into standard Datalog ensures major benefits. We can now use our FS constructs in recursive programs with the assurance of formal semantics (model-theoretic, proof-theoretic and least-fixpoint). Beyond that, we can use negation defined through the closed world assumption, and the formal theory of programs with negated goals, starting from stratified negation and proceeding to stable model semantics. Furthermore efficient implementation methods are not far behind: as we will briefly discuss later, techniques such as differential fixpoint, and magic set methods can be easily extended to Datalog^{FS}.

As we shall see later, the rewriting of Datalog^{FS} into Datalog that was used for defining its declarative semantics will not be used in its actual implementation: we will instead use a more direct approach that avoids the complexities and inefficiencies of the rewritten rules, while producing the same results.

5.4.2 Stratified Datalog^{FS}

Stratified negation extends naturally to Datalog^{FS}. Also, the rewriting rules previously defined make it possible to use variables rather than constants in the specification of FS goals. This is useful in many situations.

For instance, to find the actual number of sons that a person has, we can write:

Example 5.4. How many sons does a person have?

$$\begin{aligned} \text{csons}(\text{Name}, N) \leftarrow \text{person}(\text{Name}), N: [\text{son}(\text{Name}, _)], \\ N1 = N + 1, \neg N1: [\text{son}(\text{Name}, _)]. \end{aligned}$$

Thus `csons` must belong to a stratum that is strictly higher than `son`, whereas with respect to `person` it could be in the same stratum or in the one above it. The need to find the maximum value satisfying a FS-test clause is so common that Datalog^{FS} provides the special construct `=!` to express it. Thus the previous rule can also be expressed as:

Example 5.5. How many sons does a person have?

$$\text{csons}(\text{Name}, N) \leftarrow \text{person}(N), N =! [\text{son}(\text{Name}, _)].$$

Now, `=!` will be called the exact frequency construct and its semantics is defined by the rewriting exemplified by the rewriting of Example 5.5 into Example 5.4. In terms of concrete semantics, however, the exact frequency construct will be given a more direct and efficient implementation.

5.4.3 Recursive Datalog^{FS}

A first example illustrating the uses of Datalog^{FS} is based on an example by Ross and Sagiv [RS97]:

Example 5.6. Some people will come to the party for sure. Others will also come once they learn that three or more of their friends will come.

$$\begin{aligned} \text{willcome}(X) &\leftarrow \text{sure}(X). \\ \text{willcome}(Y) &\leftarrow 3: [\text{friend}(X, Y), \text{willcome}(X)]. \end{aligned}$$

Another interesting example is the Apriori computation. Say that our database represents market-basket data as

$$\text{baskets}(\text{basketNo}, \text{ListofItems})$$

where the items in `ListofItems` are kept in lexicographical order. For instance, “`baskets(b467, [beer, cereals, diapers])`.” could be one such fact. Then the rules below compute the list of items with frequency support of, say, 35:

Example 5.7. The APRIORI computation of item sets with frequency support 35.

$$\begin{aligned} \text{freq}(1, [\text{Itno}]) &\leftarrow 35: [\text{baskets}(\text{BID}, \text{List}), \text{in}([\text{Itno}], \text{List})]. \\ \text{cand}(2, [A, B]) &\leftarrow \text{freq}(1, [A]), \text{freq}(1, [B]), A < B. \\ \text{cand}(N1, \text{ML}) &\leftarrow \text{freq}(N, [A|L]), \text{freq}(N, [B|L]), A < B, \\ &\quad \text{ML} = [A|[B|L]], M = N - 1, \\ &\quad M: [\text{freq}(N, L1), L1 \neq [A|L], L1 \neq [B|L], \\ &\quad \quad \text{in}(L1, \text{ML})], N1 = N + 1. \end{aligned}$$

$$\begin{aligned} \text{freq}(N1, \text{ML}) &\leftarrow \text{cand}(N1, \text{ML}), \\ &\quad 35: [\text{baskets}(\text{BID}, \text{List}), \text{in}(\text{ML}, \text{List})]. \end{aligned}$$

$$\begin{aligned} \text{in}([], L). \\ \text{in}([A|L], [A|List]) &\leftarrow \text{in}(L, List). \\ \text{in}([B|L], [A|List]) &\leftarrow A < B, \text{in}([B|L], List). \end{aligned}$$

Therefore, our first rule finds frequent items with the help of the predicate $\text{in}(L1, L2)$ which is true whenever $L1$ is a sublist of $L2$: the rules implementing $\text{in}(L1, L2)$ exploit the fact that $L1$ and $L2$ are sorted. The next two rules in Example 5.7 generate candidate itemsets. Candidate pairs are simply obtained by combining singleton sets. Then, as we move to longer patterns, to construct candidate patterns of size $N + 1 \geq 3$ we combine two patterns of size N that are identical except for their lead item into the pattern ML and verify that ML contains $M = (N + 1) - 2 = N - 1$ subpatterns of size N that are frequent. Finally, we go back to our fact base to test which of the ML patterns so obtained have frequency support of 35 or better.

5.5 Multi-Occuring Predicates

In the examples considered so far, base predicates and derived predicates were always counted as providing support of one. However, there are numerous examples

where it is desirable that we specify that certain predicates should be counted as providing a support level greater than one. For instance, we might use the following representation to denote that the paper with DBLP identifier “MousaviZ11” has currently 6 references:

$$\text{ref}(\text{“MousaviZ11”}):6.$$

Naturally, $\text{Pno} = \text{“MousaviZ11”}$ now contributes with a count of six to the b-expression of the following rule:

Example 5.8. Total reference count for an author.

$$\text{tref}(\text{Author}):N \leftarrow N: [\text{author}(\text{Author}, \text{Pno}), \text{ref}(\text{Pno})].$$

The clauses “:6” and “:N” respectively used in the above fact and rule head will be called *frequency assert clauses*. The semantics of programs P with frequency assert clauses is defined by *expanding* its \bar{P} equivalent, which is obtained as follows:

- i Each rule in P having head $q(X_1, \dots, X_n):K$ and body Body is replaced by

$$\bar{q}(X_1, \dots, X_n, J) \leftarrow \text{lessthan}(J, K), \text{Body}.$$

where $\text{lessthan}(J, K)$ is a distinguished predicate used to generate all positive integers up to K , included.

$$\text{lessthan}(1, K) \leftarrow K \geq 1.$$

$$\text{lessthan}(J_1, K) \leftarrow \text{lessthan}(J, K), K > J, J_1 = J + 1.$$

- ii Each occurrence of $q(X_1, \dots, X_n)$ in the body of rules of the program so obtained is replaced by: $\bar{q}(X_1, \dots, X_n, J)$

Thus, the meaning of our program,

$$\text{ref}(\text{“MousaviZ11”}):6.$$

$$\text{tref}(\text{Author}):N \leftarrow N: [\text{author}(\text{Author}, \text{Pno}), \text{ref}(\text{Pno})].$$

is defined by its expansion into the following program:

$$\bar{\text{ref}}(\text{“MousaviZ11”}, J) \leftarrow \text{lessthan}(J, 6).$$

$$\bar{\text{tref}}(\text{Author}, J) \leftarrow N: [\text{author}(\text{Author}, \text{Pno}), \bar{\text{ref}}(\text{Pno}, J)].$$

As a result of this expansion, Example 5.8, “MousaviZ11” contributes with number 6 to the reference count for each of its authors.

If $\text{staff}(\text{Lab}, \text{Name})$ describes the current staff in each lab, the following rule can be used to determine the total number of references pointing to (papers by researchers in) each lab.

$$\text{labref}(\text{Lab}):Tot \leftarrow Tot: [\text{staff}(\text{Lab}, \text{MTS}), \text{totalref}(\text{MTS})].$$

In our previous examples, the frequency assigned to the head has the same value as the FS test clause in the body, but this needs not to be always the case. For instance, if

we estimate that, on the average, every paper authored by an MTS in a particular lab is co-authored by another MTS in the same lab, then a better estimate of the papers published by the lab will be $\text{Tot} \div 2$. Thus a better estimate of a pseudo H-factor for each lab is:

Example 5.9. Pseudo H-factors for Laboratories.

$$\begin{aligned} \text{labref}(\text{Lab}):H &\leftarrow \text{Tot}:[\text{staff}(\text{Lab}, \text{MTS}), \text{totalref}(\text{MTS})], \\ H &= \text{Tot} \div 2. \end{aligned}$$

An important property of frequency statements is that, when multiple statements hold for the same fact (base fact or derived fact) only the largest value is significant, the others are subsumed and ignored. Thus, if the following two predicates are derived,

$$\begin{aligned} \text{ref}(\text{"MousaviZ11"}):6. \\ \text{ref}(\text{"MousaviZ11"}):4. \end{aligned}$$

the second fact carries no additional information and can be simply dropped. This property leads to important applications, discussed below. However, if instead of the maximum frequency we would like to take the sum of frequencies, we can simply add an additional argument. Indeed the following two facts imply a total count of 10.

$$\begin{aligned} \text{ref}(\text{"MousaviZ11"}, \text{aaa}):6. \\ \text{ref}(\text{"MousaviZ11"}, \text{bbb}):4. \end{aligned}$$

Bill-of-materials (BOM) applications represent a well-known example of the need for recursive queries. For instance, our database might contain records such as

$$\text{assbl}(\text{Part}, \text{Subpart}, \text{Qty})$$

which, for each part number, give the immediate subparts used in its assembly and the quantity in which they are used. For instance a bicycle has 1 frame and two wheels as immediate subparts. At the bottom of the BOM DAG, we find the basic parts that are purchased from external suppliers and provide the raw materials for our assembly. Basic parts are described by $\text{basic}(\text{Pno}, \text{Days})$ denoting the days needed to obtain a basic part. Several interesting BOM applications are naturally expressed by combining aggregates and recursion, as follows:

Example 5.10. How many basic parts does an assembled part contain?

$$\begin{aligned} \text{cassb}(\text{Part}, \text{Sub}):Qty &\leftarrow \text{assbl}(\text{Part}, \text{Sub}, \text{Qty}). \\ \text{cbasic}(\text{Pno}):1 &\leftarrow \text{basic}(\text{Pno}, _). \\ \text{cbasic}(\text{Part}):K &\leftarrow K:[\text{cassb}(\text{Part}, \text{Sub}), \text{cbasic}(\text{Sub})]. \end{aligned}$$

Of course the total count of basic parts used by a part, such as `frame`, should not be retrieved directly using a goal such as $N:[\text{cbasic}(\text{frame})]$ since this will return all the positive integers up to the max N value. A goal such as $N=[\text{cbasic}(\text{frame})]$

should be used instead inasmuch as this returns the exact count of the basic subparts for `frame`. Similar observations hold for the other examples.

Simple assemblies, such as bicycles, can be put together the very same day in which the last basic part arrived. Thus the time till delivery is the maximum number of days required for all the basic parts to be delivered:

Example 5.11. How many days till delivery?

```
delivery(Pno):Days ← basic(Pno,Days).
delivery(Part):Days ← assb(Part,Sub,_),
                    Days:[delivery(Sub)].
```

Thus, for each assembled part, we find each basic subpart along with the number of days this take to arrive. The number of days is retained but not the basic part associated with it, whereby the maximum number of days required by any assembly is computed.

Mumick, Pirahesh and Ramakrishnan proposed the following interesting application [MPR90]:

Example 5.12. Companies can purchase shares of other companies; in addition to its directly owned shares, a company A controls the shares controlled by a company B when A has a controlling majority (50%) of B's shares (in other words, when A bought B).

```
cshares(C1,C3,direct):P ← owned_shares(C2,C3,P).
cshares(C1,C3,indirect):P ← P:[bought(C1,C2),
                               cshares(C2,C3,_)].
bought(C1,C2) ← C1 ≠ C2, 50:[cshares(C1,C2)].
```

5.6 Implementation & Optimization

Datalog compilers [AOT⁺03] perform a binding passing analysis on the program to determine whether top-down methods such as magic sets are applicable, and (a) if they are not, the compiler applies the differential fixpoint (a.k.a. seminaive fixpoint) method, otherwise (b) the compiler first applies the best suitable top-down rewriting method, and then applies the differential fixpoint method to the transformed program thus generated. Modulo simple modifications, this process and methods remain valid and effective for Datalog^{FS} programs. We will now illustrate these operations with the help of our previous examples.

5.6.1 Differential Fixpoint

Say that we want to find all people who will come to the party. Then, we can ask the following query `?willcome(Name)` on the program in Example 5.6.

$$\begin{aligned} \text{willcome}(X) &\leftarrow \text{sure}(X). \\ \text{willcome}(Y) &\leftarrow 3: [\text{friend}(X, Y), \text{willcome}(X)]. \end{aligned}$$

Since *Name* is a variable, no top-down compilation method is applicable here and we simply apply the basic differential fixpoint method. The application of the differential fixpoint method begins with a *factorization* step whereby the original recursive rule is re-written by factoring out one occurrence of the b-expression, as follows:

$$\begin{aligned} \text{willcome}(Y) &\leftarrow \text{friend}(X1, Y), \text{willcome}(X1), \\ &2: [\text{friend}(X2, Y), \text{willcome}(X2), X2 \neq X1]. \end{aligned}$$

Then, the symbolic differentiation is only applied to the factored out b-expression producing

$$\begin{aligned} \delta \text{willcome}(Y) &\leftarrow \delta(\text{friend}(X1, Y), \text{willcome}(X1)), \\ &2: [\text{friend}(X2, Y), \text{willcome}(X2), X2 \neq X1]. \end{aligned}$$

Indeed, while the recursive predicate *willcome* could be viewed as occurring three times in the body of our recursive rule all the recursive goals are identical, and their order is immaterial: thus they all produce the same delta rule.

Since *friend* is not a recursive predicate, then the conjunct $\delta(\text{friend}(X1, Y), \text{willcome}(X1))$ is rewritten into $\text{friend}(X1, Y), \delta \text{willcome}(X1)$ (if *friend* were also recursive then one more rule would be needed). The meaning of the rule we have constructed is clear: We take the new $\delta \text{willcome}(X1)$ obtained in the last iteration step, and we check if the following FS goal holds:

$$2: [\text{friend}(X2, Y), \text{willcome}(X2), X2 \neq X1].$$

A more relaxed factorization is preferable for recursive rules, such as the following one from Example 5.10, where the value of the FS test clause *K* is simply transferred to the head of the rule (whereas in the previous example the existence of three distinct occurrences had to be actually tested):

$$\text{cbasic}(\text{Part}):K \leftarrow K: [\text{cassb}(\text{Part}, \text{Sub}), \text{cbasic}(\text{Sub})].$$

In the relaxed factorization, the \neq condition is omitted and the b-expression inside and outside the brackets are the same, whereby the previous rule is re-written as:

$$\begin{aligned} \text{cbasic}(\text{Part}):K &\leftarrow \text{cassb}(\text{Part}, \text{Sub}), \text{cbasic}(\text{Sub}), \\ &K: [\text{cassb}(\text{Part}, \text{Sub}), \text{cbasic}(\text{Sub})]. \end{aligned}$$

This is then compiled into a single delta rule with a single delta goal:

$$\begin{aligned} \delta \text{cbasic}(\text{Part}):K &\leftarrow \text{cassb}(\text{Part}, \text{Sub}), \delta \text{cbasic}(\text{Sub}), \\ &K: [\text{cassb}(\text{Part}, \text{Sub}), \text{cbasic}(\text{Sub})]. \end{aligned}$$

5.6.2 Magic Sets

Let us now consider the case where top-down methods can be used. For instance, to know if Tom will come to the party, we can use the goal `?willcome(tom)`. After the binding passing analysis, we can apply the magic-set method, yielding the following magic predicates:

```
m.willcome(tom).
m.willcome(X) ← 3:[m.willcome(Y), friend(X, Y)].
```

This magic rule starts with $Y = \text{tom}$ and, provided that Y has at least three friends Y , propagates the magic condition to those X friends. Now, the original goal remains unchanged `?willcome(tom)` while the original rules are re-written as follows:

```
willcome(X) ← sure(X), m.willcome(X).
willcome(Y) ← 3:[willcome(X), friend(X, Y),
                 m.willcome(Y)].
```

The first rule so obtained shows that the `sure` people who are in the magic set need to be considered. Moreover, the second rule cannot be dropped as in the case of standard right-linear rules. In fact, while the magic set rules discarded everyone that does not have three friends, the modified rules remain necessary to ensure that three at least of those remaining friends will come to the party. The standard differential fixpoint will then be applied to rules so re-written.

Consider now Example 5.10, and assume that we use the goal $N = ![\text{cbasic}(\text{frame})]$ to find out how many basic parts an assembled frame contains. After performing the binding passing analysis, our Datalog^{FS} compiler would here produce the following magic-set rules (which only propagate bound values, and thus K is not in the magic-set rules):

```
m.cbasic(frame).
m.cbsasic(Sub) ← m.cassb(Part, Sub), cbasic(Part).
```

and modified rules:

```
cbasic(Pno):1 ← basic(Pno, _), m.cbasic(Pno).
cbasic(Part):K ← K:[cassb(Part, Sub), cbasic(Sub),
                   m.cbasic(Part)].
```

Observe that the modified recursive rule must be retained since it is needed to compute the values of K . This and the previous `willcome` example illustrate that the full magic set method must be often applied once the FS test clauses are also considered in the binding passing analysis—whereas without the FS-test clauses those would have been treated as right-linear rules.

5.6.3 Avoiding Expansions

The rewriting of our FS predicates that defined their semantics could also be used for their implementation. In practice however, this naive approach is prone to major inefficiencies and must be avoided. As we shall see next, efficient implementations can be achieved via simple syntactic constraints that avoid expansions without compromising the power and usability of Datalog^{FS}.

FS-test Clauses

These are easily implemented using aggregates. The COUNT aggregate will be used when all the predicates in the b-expression are single predicates (i.e., they are not among those defined as multi-occurring via FS-assert clauses). Thus, in the first rule of Examples 5.3, we will count PnoFrom for each Pno—i.e., count PnoFrom grouped by Pno in SQL parlance. In general, the operation that a compiler/interpreter will use on single predicates is counting the occurrences of local variables grouped by the global ones. For instance, in the second rule of Example 5.3 we have a conjunct where Author is the global (i.e., group-by) variable, and Pno is the local variable whose occurrences will be counted: when the count is equal or exceeds 13, then Author is returned to the head. In the recursive rule of Example 5.6 we must instead count occurrences of (X, Y) grouped by X.

The SUM aggregate must instead be used for b-expressions containing one or more multi-occurring predicates. Take for instance Example 5.10: both `cassb` and `cbasic` are multiple-occurring predicates, `Part` is the global variable and `Sub` is the local one. Moreover, each tuple in `cassb` has a multiplicity `Qty` and each tuple in `cbasic` has multiplicity `K`. Thus for each `Part`, we must sum up the products $Qty \times K$ over all `Sub` values that satisfy the join condition, and transfer the resulting sums to the head². Likewise, in the first rule of Example 5.3, we will inspect `refer` and count the occurrences of the local variable `PnFrom` grouped by the global one `Pno`³.

FS-assert Clauses

Returning to our Example 5.10, we see that FS-assertions are simple to implement. Whenever during the computation, a new instance of `cbasic(Part1) : K1` is produced from the body of the rule, we store it, and delete previously recorded `cbasic(Part0) : K0` where `Part0 = Part1` and `K0 < K1`. Thus only the current maxima of FS values are kept. Therefore, while in rules such as the second one of Example 5.10, the body is satisfied by several values of `K`, only the max of these

² If this information were stored in SQL tables, then the head of the rule would be updated using the following information:

```
select cassb.Part, sum(cassb.Qty*cbasic.K)
  from cassb, cbaic where cassb.Sub=cbasic.Sub
 group by cassb.Part
```

³ Determining local and global variables is already part of the binding passing analysis performed by current Datalog compilers [AOT⁺03].

values is of interest since it is the one that determines the range of the FS value in the head. Therefore the very re-writing that defines the semantics of FS-assert clauses provides a built-in maximization feature, which is very useful in many optimization applications, including Example 5.11 where we want to find the maximum number of days needed for all parts to be delivered.

Example 5.9 shows that the FS values transferred from the body to the head can be mapped through functions ($\div 2$ in the example). However, these functions must map maxima into maxima: thus they must be monotonic for positive numbers. In fact, if arbitrary functions were allowed, we would normally have to enumerate all FS values that satisfy the rule arguments. Monotonicity is likewise required for boolean conditions that test conditions on FS values in the body⁴. Conditions on FS-values specified using monotonic boolean functions only need to be checked for max FS-values, rather than all the integers up to the max value. We recommend that a basic Datalog^{FS} compiler should enforce this condition but allow simple equality condition as an exception. For example, the recursive rule of Example 5.6 could have also be specified as follows:

$$\text{willcome}(Y) \leftarrow K: [\text{friend}(X, Y), \text{willcome}(X)], K = 3.$$

Our basic compiler would allow this rule, and actually implement it in the same way as that of the original Example 5.6. However a basic Datalog^{FS} compiler should reject the following rule, although in terms of abstract semantics is equivalent to the previous rule (since $K = 3$ is the only integer that satisfies those inequalities):

$$\begin{aligned} \text{willcome}(Y) \leftarrow K: [\text{friend}(X, Y), \text{willcome}(X)], \\ K * K > 8, 2 * K < 7. \end{aligned}$$

Indeed these monotonicity restrictions are only meant to simplify the task of our basic compiler, and could be relaxed in more sophisticated compilers. Therefore a basic compiler (or interpreter) only needs to be smart enough to be able to verify that all the arithmetic and boolean expressions that use FS values as their arguments are indeed monotonic w.r.t. such arguments. An error message is returned when our system is not able to make such a determination. This requirement can be easily realized in practice since even simple compilers are able to infer monotonicity for simple functions or expressions: for more complex ones, users will be allowed to declare that certain functions are monotonic.

In general, the Datalog^{FS} compiler should reject user queries that require the use of the expansion rules. Therefore, predicates such as `cbasic` in Example 5.10 and `delivery` in Example 5.11 can only be called via an exact frequency goal, or a goal that checks if they hold for a particular frequency value. In our applications, we have found that these rules are easy to live with.

In summary, we have shown that (i) the powerful techniques that were developed for deductive databases and then used for recursive SQL:2003 queries [AHV95,

⁴ A boolean function $B(T)$ is monotonic w.r.t. the integer values T whenever $B(T)$ evaluating to true implies that $B(T')$ is true for every $T' > T$.

Ram98, ZCF⁺, UW97] can now be used for Datalog^{FS}, and (ii) the need for any expansion step can be easily avoided in practical applications. The elimination of any expansion step also allows us to further extend Datalog^{FS} by scaling, as discussed in the next section.

5.7 Scaling

Although the abstract semantics of Datalog^{FS} is based on counting the repeated occurrences of predicates, i.e., on integer arithmetic, it can be generalized for floating-point computation and models in a straightforward manner. All is needed is a scaling factor that maps floating point numbers into integers. For instance the percentages commonly used in real life assume a scaling factor of 100.

For instance, returning to our party example we might want to say that our level of confidence in John coming to the party is 82%. Thus we write: `confidence("John")`: 0.82, and similar confidence rankings can be assigned to every person who could possibly attend the party. Then our program becomes:

```
confidence("John"):0.82.
%...therestofthefacts...
willcome(X):C ← C:[confidence(X)].
willcome(Y):1 ← 3:[friend(X,Y),willcome(X)].
```

Since the confidence is normally less than one, we will need to add the confidence of four or more people coming to the party before we cross the threshold of 3.

In this example we were working with percentages, and thus a scale-up factor of 100. A scale factor of 100 means that all the frequency statements will be multiplied by 100 and thus will become integers. Any constant applied to FS values will also have to be multiplied by 100. Thus the program above is viewed as a more compact statement of the following program that defines its formal semantics:

```
confidence("John"):82.
%...therestofthefacts...
willcome(X):C ← C:[confidence(X)].
willcome(Y):100 ← 300:[friend(X,Y),willcome(X)].
```

This program was obtained from the previous one by multiplying FS constants by the scaleup factor of 100. This second program defines the semantics of the first one with the provision that the FS values returned by this second one will be divided by 100.

Thus by assuming scaling, we allow the use of floating point numbers in our FS-assert and FS-test clauses. This significantly expands the application range of Datalog^{FS}. For instance, say that `arc(a, b):0.66` denotes that a trip started at point a will actually take us to point b in 66 % of the cases. Then the following program computes the probability of completing a trip from a to Y along the maximum-probability path:

Example 5.13. Maximizing the probability of reaching various nodes of the graph if we start from node *a*.

```
reach(a):1.00.
reach(Y):V ← reach(X), V:[reach(X), arc(X,Y)].
```

Thus, we reached *a* with probability 1. Then, the probability of reaching *Y* via an arc from *X* is the product of the probability of being in *X* times the probability that the segment from *X* to *Y* can be completed. In the head of the rule, we only retain the maximum *V*—i.e., we only retain the path with largest probability to succeed.

In terms of implementation it is clear that the approach and compilation techniques we used to implement Datalog^{FS} when FS values were integers, remain valid and can now be used when FS values are arbitrary positive numbers (however SUM rather than COUNT will be used in the FS-test Clauses). This follows directly from the fact that, as we have described in the last section, we avoid expansions in Datalog^{FS} implementation. This also makes the semantics of our programs independent from the scale-up factor used—modulo some round-off issues discussed next.

Consider the equivalent for the program of Example 5.13 above, when the scale-up factor is 100. This scale-up multiplies by 100 both the *reach* and *arc*, for a combined scaling of 100×100 . Therefore, we must normalize back the resulting sums by dividing the results by 100.

Thus the equivalent Datalog^{FS} program for a scale-up factor of 100 is:

```
reach(a):100.
reach(Y):W ← reach(X), V:[reach(X), arc(X,Y)],
           W = V ÷ 100.
```

This example also illustrates that an implicit round-off effect is connected with the scale-up factor. For instance if $V = 0.0048$, for a scale-up factor of 100, we obtain $W = 0.00$, whereas with scale up factor of 10,000 we obtain $W = 0.0048$. Therefore there is a round-off dependency connected with the round-off factor, and this dependency can be minimized by selecting higher scale-up factors. Thus we will assume that our Datalog^{FS} implementation can use the max scale-up factor that can be efficiently supported by the target hardware (32-bit or 64-bit). With that, round-off issues will not occur in most applications; in the few situation where they are cause for concern, they should addressed using “double precision” declarations and various numeric computation techniques that are used in everyday computing.

5.8 More Advanced Applications

In this section we discuss web-oriented applications, including social networks and page rank.

5.8.1 Diffusion Models with Datalog^{FS}

The Jackson-Yariv Diffusion Model (JYDM) [JY05] was extended to elegant logic-based formalizations [SSS10], which however can lead to inefficient (NP-hard) computations [SSS10]. We will now show that JYDM can be expressed by simple and efficient Datalog^{FS} programs. In JYDM, a set V of agents is represented by a graph $G = \langle V, E \rangle$ where E are edges that represent the relationship between two agents. Each agent has a default behavior (A) and a new behavior (B). An agent i decides on whether to adopt behavior B depending on:

- (i) A constant bc_i which quantifies the extent to which agent i is susceptible to make a change (typically, bc_i measures the reward/cost tradeoffs of making the change).
- (ii) The percentage of the neighbors of the agent that took action B (e.g., the percentage of partners of the twitter user who have forwarded this tweet).
- (iii) A function $g : \{1, \dots, |V|\} \rightarrow [0, 1]$ that modifies criterion (ii) to take into account, in addition to percentage of neighbors who took action B, their number⁵. Observe that g does not directly depend on the node, it only depends on the number of its neighbors; however, g and bc combined provide JYDM with much power and flexibility in characterizing the response w.r.t. both the numbers and percentages of neighbors agents who took action B.

Now, if B_i denotes the pressure experienced by agent i to adopt behavior B , the Jackson-Yariv diffusion model is represented by the following equations:

$$B_i = bc_i * g(\Gamma_i) * \frac{1}{\Gamma_i} * \sum_{(j,i) \in E} B_j, \forall i \in V \quad (5.1)$$

where

$$\Gamma_i = \sum_{(j,i) \in E} 1$$

is the count of the neighbors.

When a B_i crosses a threshold, then agent i switches from behavior A to behavior B . Typically, this threshold can be set to 1, given that the bc_i can be scaled up/down to match the application requirements.

The JYDM-based diffusion of retweets can easily expressed in Datalog^{FS}. We represent the twitter network using atom `followd(X, Y)` to indicate that user X is followed by user Y . Moreover, in order to apply the diffusion model, we introduce the following facts:

- `bc(X, K)` means that the node X has coefficient bc_X equal to K .
- `g(N, K)` means that the function $g(N)$ returns value K .

⁵ For instance, this function can be used to express the fact that an agent who sees, say, all his 89 partners switching to B experiences a much stronger push than another twitter user who sees his only partner moving to B (although percentage-wise the two situations are the same).

The default behavior A means that a user will not retweet, while behavior B means that the user will retweet. Now B_X represent the fact that X will retweet and we represent this by the predicate $b(X)$. We assume that there is an agent $\text{source}(X)$ who first posts the tweet and starts its diffusion.

Thus, Equation 5.1 is modelled by the following rules.

Example 5.14. Equation 5.1 in Datalog^{FS}.

$$\begin{aligned} \text{coeff}(X, C) \leftarrow & K2 = ![\text{followd}(Y, X)], \text{bc}(X, V1), \\ & g(K2, V3), C = V1 * V3 / K2. \\ \\ b(X) \leftarrow & \text{source}(X). \\ b(X) \leftarrow & \text{coeff}(X, C), K = 1/C, K: [\text{followd}(Y, X), b(Y)], . \end{aligned}$$

Note that in Equation 5.1 the product

$$bc_i * g(\Gamma_i) * \frac{1}{\Gamma_i}$$

is not recursive and depends only on the features of the node i . Therefore we compute separately this value for each user by the first rule. The first goal in this rule determines the number of neighbors of node X ; the next three goals in the rule compute the increment C , that must be added to $b(X)$ for each of its neighbors who switched to B . Since there are K of these neighbors, the test condition for setting the head $b(X)$ to true is $K \times C \geq 1$; this is equivalent to the condition $K \geq 1/C$ used in our rule (which is much preferable in terms of compilation for the reasons discussed in the last section).

The answer to the query $?b(Y)$ will thus list all the users that propagated the tweet that originated from the node specified by `source`, as illustrated by the Example 5.15 below:

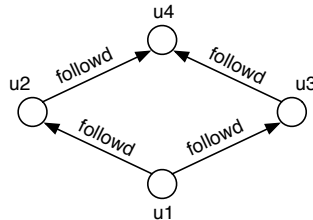


Fig. 5.1. A Diffusion Model for Twitter

Example 5.15. Consider the network in Figure 5.1 where user u_1 tweets about something. We add the following facts to previous program and compute for each user if he will retweet the post or not.

```

followd(u1, u2).   bc(u1, 1).   g(1, 1.2).   source(u1).
followd(u1, u3).   bc(u2, 0.9). g(2, 2.3).
followd(u2, u4).   bc(u3, 0.5).
followd(u3, u4).   bc(u4, 1).

```

And we obtain the following result:

$$M = \{ \dots, \text{coeff}(u_2):1.08, \text{coeff}(u_3):0.6, \text{coeff}(u_4):2.3, \\ \text{b}(u_1), \text{b}(u_2), \text{b}(u_4) \}$$

Thus, first u_2 and then u_4 will retweet u_1 's post.

5.8.2 Markov Chains with Datalog^{FS}

A Markov chain is a memory-less stochastic process that is represented by the transition matrix W of $s \times s$ components where the component w_{ij} is the probability to go from the state i to state j in one step. For each node i of the Markov chain we have: $\sum_{j=1}^s w_{ij} = 1$. A Markov chain is called *irreducible* if for each pair of states i, j , the probabilities to go from i to j and from j to i in one or more steps is greater than zero.

Computing stabilized probabilities of a Markov chain has many real-world applications, such as estimating the distribution of population in a region and determining the Page Rank of web nodes. Let P be a vector of stabilized probabilities of cardinality s , then for each component p_i of P the following property holds:

$$p_i = \sum_{j=1}^s w_{ji} \cdot p_j \quad (5.2)$$

This is the equilibrium condition that in terms of matrices can be expressed through the fixpoint equation:

$$P = W \cdot P$$

It is not a trivial task to compute this fixpoint. The simplest approach consists in assigning an initial value (e.g., 1) to all nodes and then iterating until the computation stabilizes. Unfortunately this approach could fail to converge even for irreducible chains that are guaranteed to have a non-trivial equilibrium solution. As shown in Example 5.16, below, this problem occurs even in very simple networks.

Example 5.16. Consider the simple and irreducible Markov chain in Figure 5.2. The computation of its stabilized probabilities is the following:

step	a	b	c
1 st	1.0	1.0	1.0
2 nd	0.5	2.0	0.5
3 rd	1.0	1.0	1.0
4 rd	0.5	2.0	0.5
5 rd	1.0	1.0	1.0
.	.	.	.
∞	.	.	.

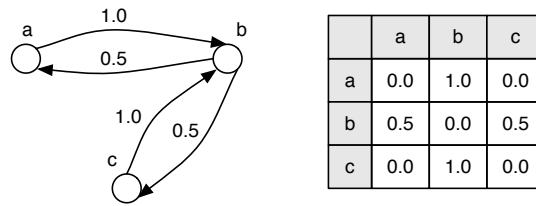


Fig. 5.2. Irreducible Markov Chain

We can see that the computation flip-flops between two states and fails to converge to the equilibrium solution.

Markov chains can be modeled quite naturally in Datalog^{FS}, and this approach provides a simple analysis and efficient algorithms for dealing with this much-studied problem.

Thus, if $p_state(X) : K$ denotes that K is the probability of node X , $1 \leq X \leq s$, and $w_matrix(Y, X) : W$ denotes that the arc from Y to X has weight W , then our Markov chain is represented by the following program:

Example 5.17. A Datalog^{FS} model for equilibrium $P = W \cdot P$ in a Markov Chain.

```

p_state(X) : K ← K : [p_state(Y), w_matrix(Y, X)].

w_matrix(1, 1) : w11.
w_matrix(1, 2) : w12.
⋮
w_matrix(s, s) : wss.

```

Thus we have obtained a positive logic program which comes endowed with well-known properties, including the fact that it defines a fixpoint equation that has one or more solutions. The least-fixpoint solution is when all $p_state(X)$ are false. This is the null solution which is not of any interest in practice, since it corresponds to every city being empty, or every page having rank 0. Thus the case of interest is when there is some non-null solution. Clearly if we take a non-null solution vector and multiply each of its components for the same constant we obtain another solution vector. We refer to this transformation as *scaling* of solutions: clearly if there is a non-null solution scaling produces an infinite number of solutions.

Now irreducible Markov chains are guaranteed to have non-null solutions; thus some of their p_state nodes are true, i.e., $p_state(X) : K$ holds for some node X and some $K > 0$ (i.e., node X has a positive solution). But since each node is reachable from every other node, via non-zero probability paths, this implies that $p_state(X) : K_x$ with $K_x > 0$ holds for every node X —i.e., our Markov chain has a positive solution at every node. Now for Markov chains that have positive solutions at every node, we can use Datalog^{FS} to express and compute efficiently such solutions. All is needed

is to state baseline facts that assert the same non-zero multiplicity for each node of our Markov chain. For instance, we can use the multiplicity of 1 and add the facts: $\text{p_state}(j)$, $j = 1, \dots, s$. These will be called *baseline* facts, and the resulting program will be called baseline program. Thus, for the example at hand we have the following Datalog^{FS} program where the three baseline facts are listed last.

Example 5.18. A baseline Program for the Markov Chain of Figure 5.3

$$\text{p_state}(X) : K \leftarrow K : [\text{p_state}(Y), \text{w_matrix}(Y, X)].$$

$$\begin{aligned} \text{w_matrix}(a, b) &: 1.0. \\ \text{w_matrix}(b, a) &: 0.5. \\ \text{w_matrix}(b, c) &: 0.5. \\ \text{w_matrix}(c, b) &: 1.0. \\ \text{p_state}(a). \\ \text{p_state}(b). \\ \text{p_state}(c). \end{aligned}$$

The least fixpoint computation for this Datalog^{FS} program is as follows:

step	a	b	c
1 st	1.0	1.0	1.0
2 nd	1.0	2.0	1.0
3 rd	1.0	2.0	1.0

Therefore, the iteration converged in three steps to a solution that is a fixpoint for the above program and captures the correct ratio between the probabilities (or the population) of the various nodes. This might in fact be all is needed in applications such as page rank where we are primarily interested in determining the rank order of pages. For other applications, we might need to normalize these results to satisfy constraints about total population or probabilities. For instance, assuming that the sum of probabilities must add up to 1, then the probability at each node can be derived using the following rule:

$$\begin{aligned} \text{p_norm}(X, \text{Pr}) \leftarrow K1 =![\text{p_state}(X)], K2 =![\text{p_state}(Y)], \\ \text{Pr} = K1/K2. \end{aligned}$$

Thus for the example at hand the normalized probability values are: $a = 0.25$, $b = 0.5$ and $c = 0.25$.

Therefore, in order to find the fixpoints for program P that models our Markov chain, we added baseline facts and obtained the bas-line program Pbl . Pbl is a Datalog^{FS} program for which we can compute the least fixpoint efficiently. Moreover every fixpoint of Pbl is also a fixpoint for P . Indeed, for any interpretation I that contains all the baseline facts, the respective immediate consequence operators produce the same results: i.e., $T_P(I) = T_{Pbl}(I)$. Therefore any fixpoint of T_P that contains all the baseline facts is also a fixpoint for T_{Pbl} and vice-versa. But since, by

its very definition, the least model of Pbl contains all the baseline facts, we have that every fixpoint for T_{Pbl} is also a fixpoint for T_P . The opposite of course is not true since the null fixpoint of T_P , and possibly others, are not fixpoint for T_{Pbl} . However, if T_P has a fixpoint that is positive at all nodes, then by multiplying the frequency at all nodes by a large enough finite constant, we obtain a fixpoint for T_P that contains all the baseline facts of T_{Pbl} . Since, for each irreducible Markov chain T_P has a finite non null fixpoint then there exists a finite fixpoint for T_{Pbl} . Therefore, the least fixpoint for T_{Pbl} is finite. That is, we can state the following theorem:

Theorem 5.19. *The least fixpoint of the baseline Datalog^{FS} program that models an irreducible Markov chain is finite.*

The operation of multiplying the frequency of the solution at each node by a positive constant is called scaling. Now since irreducible programs have positive solutions at every node we can state the following theorem:

Theorem 5.20. *Every non-null solution of an irreducible Markov chain can be obtained by scaling the least fixpoint solution of its baseline Datalog^{FS} model.*

In summary, while there has been a significant amount of previous work on Markov chains, the use of Datalog^{FS} has provided us with a simple solution for all irreducible Markov chains. In fact this solution approach holds for all Markov chains that have positive solutions at every node, and therefore for all irreducible Markov chains.

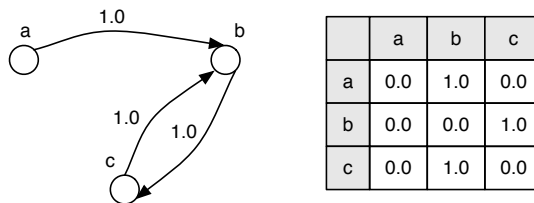


Fig. 5.3. A Reducible Markov Chain

Obviously there are many Markov chains that are not irreducible, as the one in Figure 5.3 where, using 0.1 as base line, the computation of the least fixpoint is as follows:

step	a	b	c
1 st	0.1	0.1	0.1
2 nd	0.1	0.2	0.1
3 rd	0.1	0.2	0.2
4 rd	0.1	0.3	0.2
5 rd	0.1	0.3	0.3
.	.	.	.
∞	0.1	∞	∞

Thus, the computation only converges at the first ordinal, producing infinite values and the corresponding least fixpoint is not finite. A practical solution for these situations, which is used e.g., in the Page Rank algorithm, is to introduce a dumping factor that transforms the chain in an irreducible chain. We are also investigating a different approach, where instead of modifying the original network configuration we use techniques inspired by the computation of greatest fixpoints. This research line is still at its initial stage and will be discussed in future reports.

Conclusions

In this thesis we considered the count constraints in a generative way, i.e. we studied how it is possible to materialize data that satisfy them, and we proposed some contexts in which the count constraints are meaningful. As first context, we consider the inverse frequent set mining problem. Initially, under the assumption that the itemset not in S must have support equal to zero, we defined IFM_S in which the transactions to be added in the output database must be picked from the given set S . Motivated by the intractability (NP-complete) of the problem, we proposed two heuristic approaches that have been tested with a thorough experimental activity over both synthetic and real data with good results. However, since there exist some cases in which this assumption is not valid, we provided a more general formulation, called $\kappa\text{-IFM}_{S'}$, that is able to control the supports of itemset not in S by a unique maximal threshold support for all itemset not in S . Despite this generalization is proven to be NEXP-complete, we provided an effective and efficient algorithm based on techniques of linear programming. The problem is modeled by a integer linear program with an exponential number of variables that has the property of being efficiently solved with column generation techniques. finally, we conduct an intensive activity of experimentation that highlights the quality and the efficiency of our algorithm. Interesting applications of IFM are benchmark database generation and privacy preserving data publishing. For the first one, consider a scenario in which a data set is needed as an input of data mining applications, for analysis purposes or simply for testing the implementation of new reasoning methods. The unavailability of real data often exacerbates in the case of lowly structured multi-organization virtual enterprises where the partners will never agree to share their own information, or no tracking infrastructure exists for collecting and storing log data. In this scenario, we can apply an inverse set mining technique for generating benchmark datasets with features mined from real-world datasets. In the second application, assume you are the owner of a transaction database which is willing to make available its data to any interested party, but you are wondered how to keep private information or sensitive knowledge from being disclosed. Recent literature has proposed a solution to the data sanitization problem which is inspired to inverse frequent set mining, where the

inverse mining method is the basic technique for data reconstruction while hiding association rule.

As second context, we focused on relational databases and we introduced a new type of constraints based on first order logic. Based on this constraints, a new inverse problem, called Inverse OLAP, has been formulated that is a powerful extension of Inverse Frequent itemsets Mining: given a star schema and a number of count constraints, does exist a satisfying relation database? The new problem turns out to be NEXP-complete under various conditions: combined complexity, program complexity and data complexity. We have also shown that the setting for expressing count constraints can be used for performing aggregate data exchange. Despite the high complexity of the Inverse OLAP problem, also for data complexity, an approximate solution can be found for some cases, by adopting an extension of the technique used for κ -IFM $_{\sigma}$. Moreover we showed as the count constraints in inverse OLAP represent a step toward aggregate data exchange. In this respect, we believe that inverse OLAP can be an opportunity of research for treating data exchange with aggregate constraints.

Finally, we have considered (i) Datalog^{FS} programs as synopses of (ii) expanded Datalog programs. While (ii) tends to be large and inefficient, they provide the perfect venue for defining abstract semantics, given that they are standard Datalog programs endowed with all the very desirable formal properties of this declarative logic-based paradigm. On the other hand, (i) is perfect for operational semantics since synopses can be computed very fast using aggregates. In other words, Datalog^{FS} combines the best of the two worlds, and this mix is further enhanced by the fact that all the powerful implementation and optimization techniques of deductive databases, e.g. magic sets, remain valid and effective in Datalog^{FS}. We believe that this will promote the incorporation of Datalog^{FS} powerful constructs into past and future systems. The many examples presented in this context suggest that the application range of deductive databases has been significantly broadened. Furthermore, the treatment of Markov Chains demonstrates that, by providing a simple and effective model for the analysis and solution of complex problems, Datalog^{FS} can also enrich different application domains. This opportunity provides a natural topic for future research.

References

- [AA01] D. Agrawal and C. Agrawal. On the design and quantification of privacy preserving data mining algorithms. In *Proc. 20th Symposium on Principles of Database System (PODS)*, 2001.
- [ABC⁺11] Foto N. Afrati, Vinayak R. Borkar, Michael J. Carey, Neoklis Polyzotis, and Jeffrey D. Ullman. Map-reduce extensions and recursive queries. In *EDBT*, pages 1–8, 2011.
- [ABE⁺99] M. Atallah, E. Bertino, A. Elmagarmid, M. Ibrahim, and V. Verykios. Disclosure limitation of sensitive rules. In *Proceedings of the IEEE Knowledge and Data Engineering Exchange Workshop*, pages 45–52, 1999.
- [AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [AK08] Foto N. Afrati and Phokion G. Kolaitis. Answering aggregate queries in data exchange. In *PODS*, pages 129–138, 2008.
- [AOT⁺03] Faiz Arni, KayLiang Ong, Shalom Tsur, Haixun Wang, and Carlo Zaniolo. The deductive database system ldl++. *TPLP*, 3(1):61–94, 2003.
- [APR11] Marcelo Arenas, Jorge Pérez, and Juan L. Reutter. Data exchange beyond complete data. In *PODS*, pages 83–94, 2011.
- [AS00] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM SIGMOD*, pages 439–450, 2000.
- [BV90] Catriel Beeri and Moshe Y. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *J. of the ACM*, 37:15–46, 1990.
- [Cal04] Toon Calders. Computational complexity of itemset frequency satisfiability. In *PODS*, pages 143–154, 2004.
- [Cal07] Toon Calders. The complexity of satisfying constraints on databases of transactions. *Acta Inf.*, 44(7-8):591–624, 2007.
- [Cal08] Toon Calders. Itemset frequency satisfiability: Complexity and axiomatization. *Theor. Comput. Sci.*, 394(1-2):84–111, 2008.
- [CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.
- [CG03] T. Calders and B. Goethals. Minimal k-free representations of frequent sets. In *Knowledge Discovery in Databases: PKDD 2003*, 2003. Artificial Intelligence.
- [CG07] Toon Calders and Bart Goethals. Non-derivable itemset mining. *Data Min. Knowl. Discov.*, 14(1):171–206, 2007.

- [CKV90] Stavros S. Cosmadakis, Paris C. Kanellakis, and Moshe Y. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *J. of the ACM*, 37:15–46, 1990.
- [CO05] Xia Chen and Maria E. Orlowska. A further study on inverse frequent set mining. In *ADMA*, pages 753–760, 2005.
- [COL04] X. Chen, M. Orlowska, and X. Li. A new framework for privacy preserving data sharing. In *4th IEEE ICDM Workshop: Privacy and Security Aspects of Data Mining*, pages 47–56, 2004.
- [CRfB05] Toon Calders, Christophe Rigotti, and Jean François Boulicaut. A survey on condensed representations for frequent sets. In *In: Constraint Based Mining and Inductive Databases, Springer-Verlag, LNAI*, pages 64–80. Springer, 2005.
- [DDS05] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors. *Column Generation*. Springer, 2005.
- [dMMAG] Oege de Moor Molham Aref and Georg Gottlob. Datalog 2.0: The resurgence of datalog in academia and industry. In *Proceedings of the March 2010 Proceeding held at Magdalen College, Oxford*.
- [DN03] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proc. 22nd Symposium on Principles of Database Systems(PODS)*, pages 211–222, 2003.
- [DT03] G. B. Dantzig and M. N. Thapa. *Linear Programming 2: Theory and Extensions*. Springer Series in Operations Research. Springer-Verlag, New York, 2003.
- [EGS03] A. Evfimievski, J. Gehrke, and R. SriKant. Limiting privacy breakes in privacy preserving data mining. In *Proceedings of the 22nd Symposium on Principles of Database Systems*, pages 211–222, 2003.
- [ESAG02] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–228, Edmonton, Canada, July 2002.
- [FKP05] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.
- [FKPT09] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Reverse data exchange: coping with nulls. In *PODS*, pages 23–32, 2009.
- [FPL⁺08] Wolfgang Faber, Gerald Pfeifer, Nicola Leone, Tina Dell’Armi, and Giuseppe Ielpa. Design and implementation of aggregate functions in the dlV system. *TPLP*, 8(5-6):545–580, 2008.
- [Gel93] Allen Van Gelder. Foundations of aggregation in deductive databases. In *DOOD*, pages 13–34, 1993.
- [GGZ91] Sumit Ganguly, Sergio Greco, and Carlo Zaniolo. Minimum and maximum predicates in logic programming. In *PODS*, pages 154–163, 1991.
- [GLLR07] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On reconciling data exchange, data integration, and peer data management. In *PODS*, pages 133–142, 2007.
- [GN08] Georg Gottlob and Alan Nash. Efficient core computation in data exchange. *J. ACM*, 55(2), 2008.
- [GOP11] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Ontological queries: Rewriting and optimization. In *ICDE*, pages 2–13, 2011.
- [GPSZ91] Fosca Giannotti, Dino Pedreschi, Domenico Saccà, and Carlo Zaniolo. Non-determinism in deductive databases. In *DOOD*, pages 129–146, 1991.

- [GSS09a] A Guzzo, D Saccà, and E Serra. An effective approach to inverse frequent set mining. In *Proc. of the Int. IEEE Conf. on Data Mining (ICDM'09)*, pages 806–811, 2009.
- [GSS09b] Antonella Guzzo, Domenico Saccà, and Edoardo Serra. An effective approach to inverse frequent set mining. In *ICDM*, pages 806–811, 2009.
- [GTTY06] Yuhong Guo, Yunhai Tong, Shiwei Tang, and Dongqing Yang. A fp-tree-based method for inverse frequent set mining. In *BNCOD*, pages 152–163, 2006.
- [GZ01] Sergio Greco and Carlo Zaniolo. Greedy algorithms in datalog. *TPLP*, 1(4):381–407, 2001.
- [Han05] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 2005.
- [HCXY07] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.*, 15(1):55–86, 2007.
- [Hel10] Joseph M. Hellerstein. Datalog redux: experience and conjecture. In *PODS*, pages 1–2, 2010.
- [HGL11] Shan Shan Huang, Todd Jeffrey Green, and Boon Thau Loo. Datalog and emerging applications: an interactive tutorial. In *SIGMOD Conference*, pages 1213–1216, 2011.
- [Hir10] Jorge E. Hirsch. An index to quantify an individual’s scientific research output that takes into account the effect of multiple coauthorship. *Scientometrics*, 85(3):741–754, 2010.
- [HK06] J. Han and M. Kamber. *Data mining: Concepts and techniques*. 2006.
- [JA05] Roberto J. Bayardo Jr. and Rakesh Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, pages 217–228, 2005.
- [Jea05] Daniel R. Jeske and et al. Generation of synthetic data sets for evaluating the accuracy of knowledge discovery systems. In *KDD*, pages 756–762, 2005.
- [JX08] Pawel Jurczyk and Li Xiong. Privacy-preserving data publishing for horizontally partitioned databases. In *Proceeding of the 17th ACM conference on Information and knowledge management, CIKM '08*, pages 1321–1322, 2008.
- [JY05] Matthew O. Jackson and Leat Yariv. Diffusion on social networks. *Economie Publique*, 2005.
- [KDD] KDDCUP2000. <http://www.ecn.purdue.edu/kddcup>.
- [KP88] Phokion G. Kolaitis and Christos H. Papadimitriou. Why not negation by fix-point? In *PODS*, pages 231–239, 1988.
- [LCG⁺09] Boon Thau Loo, Tyson Condie, Minos N. Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghuram Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, 2009.
- [Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
- [LKT08] Li Liu, Murat Kantarcioglu, and Bhavani M. Thuraisingham. The applicability of the perturbation based privacy preserving data mining for real-world data. *Data Knowl. Eng.*, 65(1):5–21, 2008.
- [LL03] Mark Levene and George Loizou. Why is the snowflake schema a good data warehouse design? *Information Systems*, 28:225–240, 2003.
- [LLM98] Georg Lausen, Bertram Ludäscher, and Wolfgang May. On active deductive databases: The statelog approach. In *Transactions and Change in Logic Databases*, pages 69–106, 1998.
- [Llo87] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

- [Mie03] T. Mielikainen. On inverse frequent set mining. In IEEE Computer Society, editor, *Proc. of 2nd Workshop on Privacy Preserving Data Mining (PPDM)*, pages 18–23, 2003.
- [MPR90] Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The magic of duplicates and aggregates. In *VLDB*, pages 264–277, 1990.
- [NLO06] J. Natwichai, X. Li, and M. Orłowska. A reconstructionbased algorithm for classification rules hiding. In *Seventeenth Australasian Database Conf. (ADC'06)*, pages 49–58, 2006.
- [OZ03] S. Oliveira and . Zaiane. Protecting sensitive knowledge by data sanization. In *Proceedings of the 3rd IEEE Intenational Conference on Data Mining*, pages 211–218, Melbourne, Florida, Nov 2003.
- [Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.
- [PY86] Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.
- [Ram98] Raghu Ramakrishnan. *Database Management Systems*. WCB/McGraw-Hill, 1998.
- [RH02] S. Rizvi and J. Haritsa. Maintaning data privacy in association rule mining. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 682–693, August 2002.
- [RMZ03] Ganesh Ramesh, William Maniatty, and Mohammed Javeed Zaki. Feasible itemset distributions in data mining: theory and application. In *PODS*, pages 284–295, 2003.
- [Ros06] Riccardo Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In *PODS*, pages 356–365, 2006.
- [RS97] Kenneth A. Ross and Yehoshua Sagiv. Monotonic aggregation in deductive database. *J. Comput. Syst. Sci.*, 54(1):79–97, 1997.
- [SSS10] Paulo Shakarian, V. S. Subrahmanian, and Maria Luisa Sapino. Using generalized annotated programs to solve social network optimization problems. In *ICLP (Technical Communications)*, pages 182–191, 2010.
- [SVC01] Y. Saygin, V. Verykios, and C. Clifton. Using unknowns to prevent discovery of association rules. *Sigmod Record*, 30(4):45–54, 2001.
- [TF09] Ajay Kumar Tanwani and Muddassar Farooq. Performance evaluation of evolutionary algorithms in classification of biomedical datasets. In *GECCO '09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*, pages 2617–2624, 2009.
- [UW97] Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems*. Prentice-Hall, 1997.
- [Var82] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146, 1982.
- [WHH00] Ke Wang, Yu He, and Jiawei Han. Mining frequent itemsets using support constraints. In *VLDB*, pages 43–52, 2000.
- [WW05] Yongge Wang and Xintao Wu. Approximate inverse frequent itemset mining: Privacy, complexity, and approximation. In *ICDM*, pages 482–489, 2005.
- [WWWL05] Xintao Wu, Ying Wu, Yongge Wang, and Yingjiu Li. Privacy aware market basket data set generation: A feasible approach for inverse frequent set mining. In *Proc. 5th SIAM International Conference on Data Mining*, 2005.
- [Zan11] Carlo Zaniolo. The logic of query languages for data streams. In *Logic and Databases 2011. EDBT 2011 Workshops*, pages 1–2, 2011.

- [ZAO93] Carlo Zaniolo, Natraj Arni, and KayLiang Ong. Negation and aggregates in recursive rules: the ldl++ approach. In *DOOD*, pages 204–221, 1993.
- [ZCF⁺] Carlo Zaniolo, Stefano Ceri, Christos Faloutsos, Richard T. Snodgrass, V. S. Subrahmanian, and Roberto Zicari. *Advanced Database Systems*. Morgan Kaufmann.
- [ZKM01] Zijian Zheng, Ron Kohavi, and Llew Mason. Real world performance of association rule algorithms. In *KDD*, pages 401–406, 2001.
- [ZO97] Xubo Zhang and Z. Meral Ozsoyoglu. Implication and referential constraints: A new formal reasoning. *IEEE Trans. on Knowledge and Data Engineering*, 9:894–910, 1997.

