Model-independent bounding of the supports of Boolean formulae in binary data

Artur Bykowski[1], Jouni K. Seppänen[2], and Jaakko Hollmén[2]

[1] LISI, INSA-Lyon, Bât. Blaise Pascal, 20, ave A. Einstein, F-69621 Villeurbanne Cedex, France, Artur.Bykowski@insa-lyon.fr

[2] Helsinki University of Technology, Laboratory of Computer and Information Science, P.O. Box 5400, 02015 HUT, Finland
Jouni.Seppanen@hut.fi, Jaakko.Hollmen@hut.fi

Abstract. Data mining algorithms such as the Apriori method for finding frequent sets in sparse binary data can be used for efficient computation of a large number of summaries from huge data sets. The collection of frequent sets gives a collection of marginal frequencies about the underlying data set. Sometimes, we would like to use a collection of such marginal frequencies instead of the entire data set (e.g. when the original data is inaccessible for confidentiality reasons) to compute other interesting summaries. Using combinatorial arguments, we may obtain tight upper and lower bounds on the values of inferred summaries. In this paper, we consider a class of summaries wider than frequent sets, namely frequencies of arbitrary Boolean formulae. Given frequencies of a number of any different Boolean formulae, we consider the problem of finding tight bounds on the frequency of another arbitrary formula. We give a general formulation of the problem of bounding formula frequencies given some background information, and show how the bounds can be obtained by solving a linear programming problem. We illustrate the accuracy of the bounds by giving empirical results on real data sets.

1 Introduction

Database management systems allow querying extensional data and intentional data. From the user's point of view, extensional data are the data explicitly input by the user into the system. Intentional data are not put in by the user—that information is derived from extensional data.

In *inductive* database management systems the intentional data are usually quite complex from various points of view and require a high computational effort to get them. In case of typical patterns (frequent sets, association rules), the common problem is that the domain of patterns is prohibitively large and the inductive database management system cannot compute them all. The typical approach is to let the user guide the system to the interesting patterns interactively, e.g., through queries, limiting the search space to be considered.

Then, the question is if the result of one query could be reused at least partly for obtaining the next result in an alternative way to re-computing the whole answer from extensional data. The following, more difficult question is whether there might be some summaries that the inductive database management system can gather off-line before the mining process, to improve the on-line behavior of most mining processes. Typically, one would be interested in sufficient statistics, i.e., summaries that can be substituted for the whole extensional data to avoid repetitive probing of the selection predicate for all candidate patterns. On the other hand, we prefer gathering summaries that are known to be efficient to obtain.

In this paper we pinpoint how to take advantage in a particular context of a collection of summary queries that have been evaluated against the extensional data to bound the value of the evaluation functions of other queries. Providing bounds may be interesting when we have thresholds on the evaluation function, and a tight bound can enable us to make a correct decision about accepting or rejecting a pattern in the query answer. We focus on the context of reusing previous queries (without pre-selecting) and leave open the question of choice of which summaries could be used beforehand.

Several data mining algorithms can be used for efficient computation of a large number of summaries from data. Such methods include Apriori-type algorithms for finding frequent sets [AMS$^+$96] or episodes [MTV97] in binary or sequential data and methods for clustering large data sets [ZRL97]. The summary information given by such algorithms can then be used as an efficient condensed representation of the data set. When the available summaries are orders of magnitude smaller than the data set itself (typical in case of huge data sets in data mining contexts), it could be worth using them instead of the entire data set to compute other interesting summaries. Typically, the information contained in a collection of summaries will not be sufficient to compute the precise value of all other summaries, but at least bounds could be inferred. If the accuracy of the estimated result is not enough, the partial quantitative information (bounds) can be used to better optimize the query execution plan (of a query to the original data set).

An interesting fundamental question is: how much information about the underlying data set does a collection of summaries give? In this paper we consider this question in the setting of frequent sets for binary data. Information of frequencies of different itemsets can have strong implications for the frequencies of other itemsets. For example, if we know that $f(AB) = f(A)$, then we know that $f(XA) = f(XAB)$ for any set X, a result that has been shown to be surprisingly useful in the context of so-called closures [PBTL99] and free sets [BBR00,BR01]. Also, we know that the frequency $f(X)$ of an itemset X is bounded from above by the minimum support $f(Y)$ of a subset Y of X.

More generally, if we possess the information about the frequencies of some Boolean formulae (frequent itemsets being a particular case), the frequency of any other Boolean formula can be inferred, to some extent. The main question we pose in this paper is how we could efficiently construct upper and lower bounds for the frequencies of any Boolean formula, given the existing information? We show that this formula bounding problem can in fact be solved by transforming the question into a linear program and solving that problem. In the worst case the transformation leads to a program of exponential size, but we also give empirical results showing that the transformation in many cases is an efficient one.

The paper is organized as follows. In Section 2 we define the basic notions and the support bounding task. Section 3 gives the solution, and Section 4 describes the empirical results. Finally, in Section 5, we summarize the paper and discuss open problems.

2 Problem: support queries in databases

Given a relation schema X, a family Φ of Boolean formulae over X and one more formula ψ over X, the task is to say as much as possible about the support of ψ in a relation r over X, when all we know about r are the supports of the formulae in Φ. We denote the support of a Boolean formula ψ in relation r by $[\psi]_r$, or simply $[\psi]$ when r is clear from context. The support is a number from 0 to 1.

In practice, we may also have information in the form of lower and upper bounds of some supports. For example, when mining frequent itemsets, we know that the support of any infrequent set lies beneath the frequency threshold.

We denote by $\mathrm{Int}_{\mathbb{Q}}(0,1)$ the set of all closed intervals in $[0,1] \cap \mathbb{Q}$, where $\mathbb{Q}$ is the set of rational numbers. Hereafter, we write $[a,b]$ for closed intervals of rational numbers to avoid the cumbersome notation $[a,b] \cap \mathbb{Q}$. For example, $[0.1, 0.5] \in \mathrm{Int}_{\mathbb{Q}}(0,1)$. Singletons are special cases of intervals; if we know that the support of a formula θ is exactly 0.3, we denote this by $[\theta] \in [0.3, 0.3]$. The reason to limit the support bounds to rational numbers is that the support of any Boolean formula is always rational, since it is a multiple of $1/|r|$.

Definition 1. *The* formula bounding task $\textsc{Bound}(X, \Phi, f, \psi)$ *has the following components: a relation schema X, a set Φ of Boolean formulae over X, a function f from Φ to* $\mathrm{Int}_{\mathbb{Q}}(0,1)$*, and a Boolean formula ψ over X. The solution of the task is the smallest set $I \subseteq [0,1]$ such that $[\psi]_r \in I$ for all relations r over X fulfilling the constraint $[\phi]_r \in f(\phi)$ for all $\phi \in \Phi$.*

In other words, we want a sound and complete inference procedure for the support bounds of Boolean formulae. We call a procedure *sound* if it doesn't rule out any possible solutions: for $q \notin I$, there should be no relation r fulfilling the constraints defined by f such that $[\psi]_r = q$. Conversely, the set I returned by a *complete* procedure is such that every solution $q \in I$ is realizable in some relation fulfilling the constraints. (A trivially complete but non-sound procedure returns $I = \emptyset$ for all inputs; the similar sound but non-complete procedure always returns $I = [0,1]$.) The problem is NP-hard, since it requires solving the satisfiability of ψ.

The reason for the name $\textsc{Bound}$ is that finding tight upper and lower bounds for the numbers in I solves the problem.

Lemma 1. *If the solution I of* $\textsc{Bound}(X, \Phi, f, \psi)$ *is nonempty, then I is an interval of rational numbers.*

Proof outline. Given two relations with different supports for a formula, we can take a suitable number of copies of each to form a larger relation where the formula has any given rational support between the two supports.

3 Solving the bounding task by linear programming

In this section, we describe a solution to the Bound task of Definition 1. The solution is based on linear programming, and it is both sound and complete.

3.1 Change of variables

Several kinds of equalities and inequalities hold in all relations. For example, $[A+B] = [A]+[B]-[AB]$ by the combinatorial inclusion-exclusion principle, and $0 \le [AB] \le [A] \le 1$ by the anti-monotonicity of support. A procedure for the Bound task has to incorporate all results of this type.

Let us analyze how these results could be proved. The inequality follows from the observation that $[A] = [AB] + [A\overline{B}]$ and that the support of $[A\overline{B}]$ lies in the interval $[0,1]$. A similar idea gives a proof of the inclusion-exclusion formula:

$$\begin{aligned}[A+B] &= [AB] + [A\overline{B}] + [\overline{A}B] \\ &= \big([AB] + [A\overline{B}]\big) + \big([AB] + [\overline{A}B]\big) - [AB] = [A] + [B] - [AB].\end{aligned}$$

This suggests that a change of variables can make the needed results simpler to prove. To that end, we make the following definitions.

Definition 2. *Given an attribute $A \in X$, the* positive literal *based on A is the Boolean formula A, and the* negative literal *based on A is the Boolean formula $\overline{A}$. A* literal *based on A is either the positive literal or the negative literal based on A.*

Definition 3. *A* clause *over the attribute set X is a conjunction of zero or more literals based on different attributes.*

Our definition of a clause allows an attribute to appear at most once, in either a negative or a positive literal. For example, $B\overline{C}\overline{E}$ is a clause over the set $\{A, B, C, D, E\}$, but $B\overline{B}\overline{E}$ and $BB\overline{E}$ are not. The true constant $\top$ is a clause as the degenerate case of zero literals.

Definition 4. *A* full clause *over the attribute set X is a clause with exactly $|X|$ literals.*

In a full clause each attribute appears exactly once, either as a negative or a positive literal. For example, the conjunction $AB\overline{C}D\overline{E}$ is a full clause over the set $\{A, B, C, D, E\}$, whereas $B\overline{C}\overline{E}$ is not. In the language of propositional logic, a full clause fully describes a model over the given attribute set.

Full clauses are important for two reasons. First, there is a natural correspondence between relations and assignments of supports to full clauses. Given

	AB	$A\overline{B}$	$\overline{A}B$	$\overline{A}\overline{B}$
$[\top] = 1.0$	$\times$	$\times$	$\times$	$\times$
$[A] = 0.6$	$\times$	$\times$		
$[B] = 0.7$	$\times$		$\times$	
$[AB] \in [0, 0.5]$	$\times$			

Table 1. Example bounding task with decomposition into full clauses

a relation r, any full clause θ over the schema of r is satisfied by some nonnegative integral number of identical tuples in r. Conversely, given an assignment of nonnegative rational supports for all full clauses summing up to 1, it is simple to construct a relation giving rise to these supports.

The second reason is that all formulae can be decomposed into full clauses (for formulae corresponding to typical queries it is easy). We record this in the following two propositions.

Proposition 1. *Every Boolean formula over an attribute set X can be equivalently written as a disjunction $C_1 + C_2 + \cdots + C_p$ of distinct full clauses C_i. We call this the* full disjunctive normal form*.*

Proposition 2. *The support of any Boolean formula ϕ over an attribute set X can be written as a sum of supports of distinct full clauses. That is, there is a set of full clauses $C_1, C_2, \ldots, C_p$ such that $[\phi]_r = [C_1]_r + [C_2]_r + \cdots + [C_p]_r$ for any relation r over X.*

These results enable us to untangle the complex interrelations of formulae. The supports of distinct full clauses are independent of each other[3], so any distribution of nonnegative supports for full clauses corresponds to a possible relation. Where the support of a Boolean formula appears in a constraint equality or inequality, we can invoke Proposition 2 to replace it by a sum of the supports of the corresponding full clauses. This amounts to a linear change of variables.

As an example, we consider an instance of $\textsc{Bound}(X, \Phi, f, \psi)$ with $X = \{ A, B \}$, $\Phi = \{ \top, A, B, AB \}$, and $\psi = AB$. After the change of variables, we have the system depicted in Table 1 which we should solve for $[AB]$. We have the additional information that $0 \le [\theta_i] \le 1$ for all formulae θ_i, but we need not worry about the inclusion-exclusion principle or similar rules. We continue this example at the end of Section 3.2.

3.2 Linear programming

We now turn to the classic optimization problem called linear programming. Our treatment is necessarily brief; see, e.g., Chapter 21 in [Kre93] for a good

[3] With the restriction that the supports of all full clauses sum up to 1; but this gives only a scaling factor.

introduction to the subject, or the Linear Programming FAQ[4] for a comprehensive list of references. For the computational complexity of linear programming, see e.g. [MSW96]; briefly, common algorithms such as Simplex tend to be useful in practice although they have worst-case exponential complexity, but more sophisticated algorithms achieve lower complexity.

Definition 5. *The linear programming problem* $\mathrm{LP}(\mathcal{A}, \mathcal{B}, \mathcal{C})$ *comprises an* $m \times n$ *matrix* $\mathcal{A}$*, an* m*-element column vector (*$m \times 1$ *matrix)* $\mathcal{B}$*, and an* n*-element row vector (*$1 \times n$ *matrix)* $\mathcal{C}$*. The solution of the problem is the vector* $\boldsymbol{x}$ *that minimizes the scalar value* $\mathcal{C}\boldsymbol{x}$ *subject to the restrictions* $\mathcal{A}\boldsymbol{x} \leq \mathcal{B}$, $\boldsymbol{x} \geq \mathbf{0}$*. We also denote by* $\mathrm{LP}'(\mathcal{A}, \mathcal{B}, \mathcal{C})$ *the problem that is otherwise similar but where the first restriction is replaced by* $\mathcal{A}\boldsymbol{x} = \mathcal{B}$*.*

The matrix $\mathcal{C}$ *is said to express the* objective function*, and* $\mathcal{A}$ *and* $\mathcal{B}$ *state the* constraints *of the problem.*

The problems $\mathrm{LP}(\mathcal{A}, \mathcal{B}, \mathcal{C})$ and $\mathrm{LP}'(\mathcal{A}, \mathcal{B}, \mathcal{C})$ are equivalent in expressive power and computational complexity. We use the first formulation in the fully general case of the formula bounding task BOUND (Definition 1). For the kinds of inputs we get from Apriori and similar procedures, we actually have equalities for all input formulae, so we can use $\mathrm{LP}'(\mathcal{A}, \mathcal{B}, \mathcal{C})$. Note that equalities $y = z$ can always be converted to the inequalities $y \leq z$ and $y \geq z$. We map the problem BOUND into an instance of a linear programming problem $\mathrm{LP}(\mathcal{A}, \mathcal{B}, \mathcal{C})$ or $\mathrm{LP}'(\mathcal{A}, \mathcal{B}, \mathcal{C})$ (depending on the kind of input). We talk about LP and inequalities in the following, but the case of LP' and equalities is similar.

Assume now that I is the solution of an instance of $\mathrm{BOUND}(X, \Phi, f, \psi)$. By Lemma 1, we know that if the set I is nonempty, it is a subinterval of $[0, 1]$ (in rationals). Therefore, we proceed to compute its infimum; the case of the supremum is symmetric. Let $n = |X|$; then there are 2^n full clauses over X.

As outlined at the end of the previous subsection, we take all the equalities and inequalities implied by f, and replace all occurrences of the supports of formulae by sums of supports of full clauses. In addition, we need to take the equality $\sum_i [\theta_i] = 1$, summing over all full clauses θ_i.

We will represent all of these equalities and inequalities in one big matrix inequality $\mathcal{A}\boldsymbol{x} \leq \mathcal{B}$. We tentatively let $\boldsymbol{x} = ([\theta_1]\,[\theta_2]\,\ldots\,[\theta_{2^n}])^{\mathrm{T}}$, the vector comprising the supports of all full clauses θ_j in some canonical order; but the optimizations described after Theorem 1 can reduce $\boldsymbol{x}$. Now every inequality of the form $k_1[\theta_1] + k_2[\theta_2] + \cdots + k_{2^n}[\theta_{2^n}] \leq z$ with integers $k_1, \ldots, k_{2^n}$ becomes the row $(k_1\, k_2\, \ldots\, k_{2^n})$ in $\mathcal{A}$ and the corresponding element z in $\mathcal{B}$. Inequalities with the $\geq$ sign can be transformed by multiplying with -1.

Having encoded all the constraints of the problem in $\mathcal{A}$ and $\mathcal{B}$, we now have to select $\mathcal{C}$ so that the solutions to the LP problem correspond to the supports of ψ. We once again invoke Proposition 2 to turn $[\psi]$ into a sum of supports of full clauses. Thus $\mathcal{C}$ will be a 0/1 vector with $\mathcal{C}\boldsymbol{x} = [\psi]$, and minimizing $\mathcal{C}\boldsymbol{x}$ subject to the constraints gives the required infimum. Linear programming yields a rational value for the infimum, since for example the Simplex algorithm [Kre93, §21.3]

[4] `http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html`

uses only sums, differences, products and ratios to solve LP. Thus, the infimum corresponds to an assignment of nonnegative rational values to the supports of the full clauses, summing to 1 and obeying all the constraints of the original problem. Multiplying all the supports by the least common multiple of their denominators gives integer counts, whence a relation can be constructed. Thus the infimum is actually a minimum.

We have now proved the following theorem.

Theorem 1. *The formula bounding task* $\textsc{Bound}(X, \Phi, f, \psi)$ *can be reduced to the linear programming task* $\mathrm{LP}(\mathcal{A}, \mathcal{B}, \mathcal{C})$*. The matrix* $\mathcal{A}$ *will have* $O(|\Phi|)$ *rows and* 2^n *columns, and the vectors* $\mathcal{B}$ *and* $\mathcal{C}$ *will both have* 2^n *elements, where* $n = |X|$*.*

One of the reasons we used full clauses was that they can be used to answer any support queries of Boolean formulae. However, many other families of formulae have this property. For example, Proposition 1 of [MT96] implies that the family of all conjunctions of atoms can be used to determine the supports of all Boolean formulae. Let us define a *representation* Θ over X as a family of formulae such that the counts of all Boolean formulae over X can be determined from the counts of the formulae in Θ. In this context, we use integer counts $\mathrm{count}_r(\theta) = \sum_{t\in r}[\theta]_t$ instead of supports $[\theta]_r = \mathrm{count}_r(\theta)/\mathrm{count}_r(\top)$.

Any representation that works *for all* r must have 2^n formulae. Indeed, given the counts corresponding to a representation, we can use Proposition 2 to form a linear system of equations from which the counts of full clauses can be solved. If there are fewer than 2^n equations, the system is underdetermined, and since all its factors are integers, it will have infinitely many integral solutions. It is therefore relatively easy to construct two relations with the same counts of all formulae of the supposed representation but different counts of some full clauses.

However, this does not rule out smaller representations that work for specific relations. When storing the counts of the conjunctions-of-atoms representation, we can leave out some counts that can be derived from others. If, e.g., there are no tuples satisfying the conjunction AB, we can leave out the count of ABC, and if the counts of D and DE are equal, we need store only one of the counts of AD and ADE. Similar ideas have been studied in [ML98,BBR00,BR01].

In our problem, we use fractional supports, not counts, which leaves one less degree of freedom. Since the supports of full clauses must add up to 1, we can leave out one number from the full-clauses representation.

In the case of LP', where $\mathcal{A}$ is a 0/1 matrix, we can often prune the problem. If some row $\boldsymbol{a}_i$ of the matrix $\mathcal{A}$ is less than or equal to another row $\boldsymbol{a}_j$, we can replace $\boldsymbol{a}_j$ by $\boldsymbol{a}_j - \boldsymbol{a}_i$, while doing the corresponding replacement in $\mathcal{B}$. Sometimes this will result in a zero in $\mathcal{B}$; we can then deduce that several unknowns are zero and prune them. Even if this doesn't occur, the matrix becomes sparser, which helps with some algorithms that solve linear programming problems.

We now continue the example bounding task of Table 1. We reduce the system depicted in the table to $\mathrm{LP}(\mathcal{A}, \mathcal{B}, \mathcal{C})$ with $\boldsymbol{x} = ([AB]\ [A\overline{B}]\ [\overline{A}B]\ [\overline{A}\overline{B}])^{\mathrm{T}}$. For example, the second equation is translated from $[AB] + [A\overline{B}] = 0.6$ to $(1\ 1\ 0\ 0)\boldsymbol{x} \leq 0.6$

and $(-1\ -1\ 0\ 0)\boldsymbol{x} \leq -0.6$. These inequalities form the third and fourth lines of $\mathcal{A}$ and $\mathcal{B}$ (see below). In this case, the first equation already forms the consistency constraint $\sum[\theta] = 1$, so we need not add it now.

We obtain the following matrices:

$$\boldsymbol{x} = \begin{pmatrix} [AB] \\ [A\overline{B}] \\ [\overline{A}B] \\ [\overline{A}\overline{B}] \end{pmatrix}, \quad \mathcal{A} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ -1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix}, \quad \mathcal{B} = \begin{pmatrix} 1 \\ -1 \\ 0.6 \\ -0.6 \\ 0.7 \\ -0.7 \\ 0.5 \\ 0 \end{pmatrix},$$

and $\mathcal{C}$=(1 0 0 0) (resp. $\mathcal{C}$=(−1 0 0 0)) for finding the lower (resp. the upper) bound of $[AB]$. Solving these two LP problems gives the minimum 0.3 (with $\boldsymbol{x} = (0.3\ 0.3\ 0.4\ 0.0)^{\mathrm{T}}$) and the maximum 0.5 (with $\boldsymbol{x} = (0.5\ 0.1\ 0.2\ 0.2)^{\mathrm{T}}$). We can obtain actual relations by multiplying the values of $\boldsymbol{x}$ by 10.

4 Experiments

We investigated the properties of the bounding procedure on two data sets. The first is *connect-4* containing some game-state descriptions, the second is *anpe*, a database about unemployed people, set up by the French unemployment agency. We describe the specific properties of the data sets along with our results in Sections 4.2 and 4.3.

We used as input to the bounding procedure different collections of frequent itemsets along with their supports [AMS$^+$96,MT96]. As explained previously, an itemset is interpreted as the Boolean conjunction of items that it contains. Different collections of frequent itemsets correspond to different support thresholds, denoted by min_{supp}.

In the implementation of the experiments, we used a less voluminous, although totally equivalent, representation of frequent itemsets, first described in [BR01]. Since this representation is smaller than all frequent itemsets, the resulting Φ contains less queries. The equivalence of representations guarantees that the same information can be inferred from it as from all frequent itemsets and their supports. We controlled the equivalence by redoing some of the experiments using the ordinary frequent itemsets, and got exactly the same results.

4.1 The framework of the experiments

We can compute the support of a Boolean formula over an itemset X exactly, if we know the supports of all subsets of X. The procedure for this computation in [MT96] is also applicable when we know the supports of frequent sets only, but then it will yield approximate bounds—it is sound but not complete. Thus, we test our new contribution using formulae over infrequent itemsets.

The protocol of the experiments can be simply put as following: we compare the average size of intervals inferred by Bound for 100 formulae, for which the combinatorial support-computing procedure of [MT96] is confronted with infrequent (thus missing) terms. The infrequent terms are due to the fact that the support threshold we use to mine frequent itemsets (considered further in the experiments with their corresponding supports as formulae with known supports) exceeds the support of some terms required by the procedure of [MT96].

The detailed protocol is the following. For each of the two data sets, we selected $k = 100$ random itemsets $X_1, ..., X_k$ that have 10 items each and whose supports do not exceed a predefined σ_{max} (10% for *connect-4* and 0.1% for *anpe*). To avoid selecting only itemsets with very low support, which typically account for the clobbering majority of all itemsets, we weighted the probability of selecting an itemset X proportionally to its support $[X]$. Even then, most of the selected itemsets have low support compared to σ_{max} (on average, 2.26% for *connect-4* and 0.010% for *anpe*).

Based on these itemsets, we randomly drew k Boolean formulae $\psi_1, ..., \psi_k$, one formula, ψ_i, over each X_i. To mimic formulae of interest in real life, for each X_i we first selected a subset $Y_i \subseteq X_i$ of items, each item of X_i with probability 0.7. Then we defined ψ_i as a disjunction of random full clauses over Y_i. We included each full clause θ in ψ_i with probability $0.5 - 0.04j$, where j is the number of negative literals in θ. Thus, we preferred clauses with more positive literals. For example, a clause with 10 negative literals had the probability of 0.1 to be included in ψ_i.

Then we computed $\textsc{Bound}(X_i, \Phi_i, f_i, \psi_i)$ where Φ_i consists of the precomputed frequent sets among the subsets of X_i, and f_i assigns to each frequent set its known support.

We report two scores, each of them is an average over the 100 computations. Denoting the resulting lower and upper bounds by L_i and U_i for each computation, the first score is the average of $U_i - L_i$, the second the average of $(U_i - L_i)/U_i$, both averages over $i \in \{1, \ldots, 100\}$.

4.2 Experiments with *connect-4*

The *connect-4* data set is very dense. It contains relatively small number of items (129) and rows (67 557).

In Figure 1 (left) we report the average size of the interval returned by the bounding procedure for different values of min_{supp}. As we can see, a lower min_{supp} results in a better bounding precision. This is due to the increasing number of input itemsets, therefore a richer information about the original data set. However, the cost associated with the computation and the use of these more voluminous collections of summaries also increases.

4.3 Experiments with *anpe*

The *anpe* data set is quite uncorrelated. With its 214 items and over 109 000 rows, it is significantly bigger than *connect-4*.

Frequent set mining extracts relatively small collections, unless we set a very small min_{supp}. We chose to extract itemsets at these low thresholds. In Figure 1

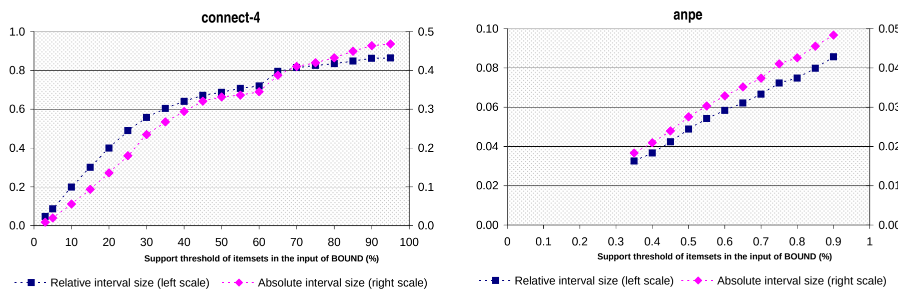

Fig. 1. Average interval size vs input itemsets' support threshold produced by BOUND on the *connect-4* (left) and *anpe* (right) data sets.

(right) we report the average interval sizes. As previously, we relate the scores to different min_{supp}.

Owing to the lower support thresholds, the resulting intervals were much smaller than in the previous experiment (i.e. the results of bounding were more precise). Observe that both the relative and the absolute errors are below 0.1 for all runs.

4.4 Observed running times

In our experiments, we first gather summary query answers, to simulate either off-line or on-the-fly collecting of highly processed information. Then, we draw random formulae, as described in Section 4.1. For each random formula, we execute two steps: conversion into an $\mathrm{LP}'(\mathcal{A}, \mathcal{B}, \mathcal{C})$ problem and solving it.

During the experiments, we observed that frequent itemset mining is the most expensive phase, despite the optimization of using an efficient condensed representation of the itemset collection described in [BR01]. For example, for the *connect-4* data set and $min_{supp} = 5\%$ it took more than 3000 seconds. Conversion to $\mathrm{LP}'(\mathcal{A}, \mathcal{B}, \mathcal{C})$ took 4.78 seconds per formula on average, and solving $\mathrm{LP}'(\mathcal{A}, \mathcal{B}, \mathcal{C})$ took only about 3.1 seconds per formula. Thus, the bounding procedure can be quite efficient in practice, after the frequent itemset mining has been performed.

5 Discussion and Future Work

We have considered the problem of bounding the support of Boolean formulae when some aggregate information is available. We showed that the bounding problem can be reduced to a linear programming problem whose size can in the worst case be exponential in the number of attributes. While our result is foremost a theoretical one, we also gave empirical results showing that the bounding method can be effectively used to obtain additional information from frequent itemsets or other summaries.

We emphasize that our aim is to find exact bounds. Another approach would be to approximate the frequency of the query and give some kind of tail bounds for the error of the approximation. The most natural way would be to take a sample from the database and compute all queries on the sample; thus, instead of frequent sets, the sample would serve as the representation of the original data. This kind of a method has been used for computing frequent sets (see [Toi96]). A more sophisticated approximation can be based on frequent sets (or similar summaries) by building a probabilistic model over the variables occurring in the formula. A method using the maximum entropy principle is described in [PMS00]. Like our solution, it suffers from exponential complexity in the number of variables occurring in the query.

Several open problems remain. One area is obtaining a faster method for the inference problem. With large, redundant summaries such as frequent itemsets, the solution by linear programming is quite slow, and it is in many cases outperformed by the simple "scan the database once and count" method. The method could, however, be useful in cases where the data set is not available or where the set of queries Φ (corresponding to known supports) carries a lot of information condensed in well chosen summaries, orders of magnitude smaller than the data set itself. Thus, the following fundamental issue is interesting.

Problem 1. Given a relation r, an amount Z of storage, and a class of queries Ψ that we wish to perform on r, what should we store in Z (which presumably cannot hold all of r) in order to most effectively answer the queries in Ψ?

Frequent sets are typically redundant collections, and thus are not optimal. It is not clear whether the less redundant AD-trees [ML98] would be optimal. If we store in Z the answers to some Boolean queries $\phi_1, \phi_2, \ldots, \phi_N$, the linear programming approach shows the limits of what we can reconstruct. Perhaps a suitable set of formulae would allow an analytical solution, possibly only approximate, of the linear program. The problem of computing frequent sets from data has been extensively studied, and they were were used in [MT96], which formed the starting point for our research. But the linear programming framework does not depend on them—it can be used with supports of any formulae.

Another interesting issue is how to relax (if possible) the requirements of Definition 1 if the complete procedure is too slow. We do not want to give unsound answers, but too wide intervals are not necessarily harmful. The simplest incomplete and sound algorithm "return the interval $[0, 1]$" is not useful, but we suspect there might be a reasonably fast compromise between it and the complete linear programming approach.

Problem 2. How close to completeness can a polynomial-time (or linear-time, or randomized polynomial-time) sound solution to Bound come?

Acknowledgements

Part of the work was done when Artur Bykowski was visiting the Laboratory of Computer and Information Science at Helsinki University of Technology. The *connect-4* data set was provided by researchers at the IBM Almaden research center, and the *anpe* data set was preprocessed and anonymized by Christophe Rigotti at INSA-Lyon. Professor Heikki Mannila asked us the question about support inference and gave useful comments on an earlier version of this manuscript.

References

[AMS$^+$96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12, pages 307–328. AAAI Press, 1996.

[BBR00] Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Approximation of frequency queries by means of free-sets. In *PKDD'00*, Lecture Notes in Computer Science, Vol. 1910, pages 75–85. Springer, 2000.

[BR01] Artur Bykowski and Christophe Rigotti. A condensed representation to find frequent patterns. In *PODS'01*, Santa Barbara, CA, USA, May 2001. ACM.

[Kre93] Erwin Kreyszig. *Advanced Engineering Mathematics.* John Wiley Inc., seventh edition, 1993.

[ML98] Andrew Moore and Mary Soon Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998.

[MSW96] Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996.

[MT96] Heikki Mannila and Hannu Toivonen. Multiple uses of frequent sets and condensed representations: Extended abstract. In *KDD'96*, pages 189–194. AAAI Press, August 1996.

[MTV97] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[PBTL99] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, 1999.

[PMS00] Dmitry Pavlov, Heikki Mannila, and Padhraic Smyth. Probabilistic models for query approximation with large sparse binary datasets. In *Proc. UAI-00*, 2000.

[Toi96] Hannu Toivonen. Sampling large databases for association rules. In *Proc. VLDB*, pages 134–145, 1996.

[ZRL97] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.