Towards Efficient Metaquerying

Rachel Ben-Eliyahu - Zohary
Communication Systems Engineering
Ben-Gurian University of the Negev
Beer-Sheva
Israel
rachel@bgumail.bgu.ac.il

Ehud Gudes
Mathematics and Computer Science
Ben-Gurion University of the Negev
Beer-Sheva
Israel
ehud@cs.bgu.acM

Abstract

Metaquery (also known as *metapattem)* is a datamining tool useful for learning rules involving more than one relation in the database. A metaquery is a template, or a second-order proposition in a language L, that describes the type of pattern to be discovered. This tool has already been successfully applied to several real-world applications.

In this paper we advance the state of the art in metaqueries research in several ways. First, we analyze the related computational problem and classify it as NP-hard, with a tractable subset that is quite immediately evident. Second, we argue that the notion of *support* for metaqueries, where support is intuitively some indication to the relevance of the rules to be discovered, is not adequately defined in the literature, and propose our own definition. Third, we propose some efficient algorithms for computing support and present preliminary experimental results that indicate that our algorithms are indeed quite useful.

1 Introduction

With the tremendous growth in information around us, datamining is emerging as a vital research area among the AI and Databases communities [Fayyad *et* a/., 1996]. Metaquerying [Shen, 1992; Wei-Min Shen and Zaniolo, 1996] is a very promising approach for datamining in relational or deductive databases. Metaqueries serve as a generic description of the class of patterns to be discovered and help guide the process of data analysis and pattern generation. Unlike many other discovery systems, patterns discovered using metaqueries can link information from many tables in databases. These patterns are all relational, while most machine-learning systems can only learn prepositional patterns.

Metaqueries can be specified by human experts or alternatively, they can be automatically generated from the database schema. Either way, they serve as a very important interface between human "discoverers" and the discovery system.

student	course
Ron	Calculus1
Dana	Algebra
Ana	Intro to CS

course	department
Calculus1	mathematics
Algebra	mathematics
Intro to CS	computer science

student	department
Ron	mathematics
Dana	mathematics
Ana	computer science

Figure 1: The relations student-course, course-department, and student-department

A metaquery R has the form

$$T \longleftarrow L_1, ..., L_m \tag{1}$$

where T and L_i are literal schemas. Each literal schema L_i has the form $Q_i(Y_1, ..., Y_{n_i})$ where all non-predicate variables Y_k are implicitly universally quantified. The expression $Q_i(Y_1, ..., Y_{n_i})$ is called a *relation pattern.* The right-hand-side $L_1, ..., L_m$ is called the *body* of the metaquery, and T is called the *head* of the metaquery. The predicate variable Q_i can be instantiated only to a predicate symbol of the specified arity n_i. The instantiation must be done in a way which is consistent with the variable names.

For example, suppose that you have a database having the relations depicted in Figure 1. Let P,Q, and R be variables for predicates, then the metaquery

$$R(X, Z) \longleftarrow P(X, Y), Q(Y, Z) \tag{2}$$

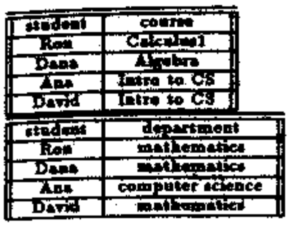
specifies that the patterns to be discovered are transitivity relations

$$r(X, Z) \longleftarrow p(X, Y), q(Y, Z)$$

where p,g, and r are specific predicates. One possible result of this metaquery on the database in Figure 1 is the pattern

$$\textbf{stud-dep}(X, Z) \longleftarrow \textbf{stud-course}(X, Y), \textbf{course-dep}(Y, Z) \tag{3}$$

which means intuitively that if a student takes a course from a certain department then he must be a student of that department.



student	course
Ron	Calculus1
Dana	Algebra
Ana	Intro to CS
David	Intro to CS

student	department
Ron	mathematics
Dana	mathematics
Ana	computer science
David	mathematics

Figure 2: The new relations student-course and student-department

1.1 The notion of *support* and *confidence* for metaqueries

Suppose that we are given the metaquery (2) again, but instead of the relations shown in Figure 1, we have the relations shown in Figure 2 (the relation course-department is the same as in Figure 1).

The rule (3) doesn't hold in all cases anymore. But we can say that it holds in 75% of the cases, or in other words, that it has *confidence* 0.75.

Let us now consider another set of relations from an employee database of an Israeli high-tech company having 1000 employees. The company is located in Beer-Sheva, and all the employees except Ana live in Beer-Sheva. Ana lives in Kibutz Shoval near by. None of the employees, except Guy, where born in the area. Guy was born in Kibutz Shoval and Ana is the boss of Guy.

Now suppose that we pose the following metaquery:

$$R(X,Y) \longleftarrow P(X,Z), Q(Y,Z) \tag{4}$$

An answer to this metaquery can be:

$$\mathbf{boss}(X,Y) \longleftarrow \mathbf{empl\text{-}born}(X,Z), \mathbf{empl\text{-}lives}(Y,Z) \tag{5}$$

with confidence 1.0. So we are going to learn the rule that if an employee was born in some place and a second employee lives in that same place, then the first employee must be the boss of the second. This rule is useless because it is based on a very weak evidence two people out of the 1000 who work in this company. Rules in which the rule body is satisfied by only a very low fraction of the relations involved should be avoided, or in other words, we would like rules with high *support.*

Hence, each answer to a metaquery is a rule accompanied by two numbers: the *support* and the *confidence.* The threshold for the support and confidence is provided by the user. Intuitively, the *support* indicates how frequently the body of the rule is satisfied, and the *confidence* indicates what fraction of the tuples which satisfy the body also satisfy the head. Similar to the case of association rules, the notions of *support* and *confidence* have two purposes: to avoid presenting negligible information to the user and to cut off the search space by early detection of low support and confidence.

Formally, given a rule

$$t \longleftarrow r_1, \ldots, r_m, \tag{6}$$

let *J* denote the relation which is the equijoin of $r_1, \ldots, r_m,$ and let *Jt* be the relation which is the eqinjoin[1] of *J* and *t.* Where *y* and *x* are some relations, let y_x, be the projection of *y* over the fields which are common to *y* and *x,* and let $|x|$ be the number of tuples in *x.* For each i, i = l...m define S_i to be the fractio$\frac{|J_{r_i}|}{|r_i|}$. The *support* of the rule (6) is the maximum over S_i, t = l...m. Less formally, the support is the maximum fraction of any relation r_i in J. The *confidence* of (6) is the fraction of t that appears in J, or, formally, the *confidence* of (6) is $\frac{|Jt_J|}{|J|}$.

The support that we have defined has the following useful property:

Claim 1.1 *For any two relations r_i and r_j in the rule (6) that have at least one common attribute variable, S_i is not bigger the the fraction of r_i that participates in the equijoin between r_i and r_j*

This property enables us to get an upper bound on the support of the rule (6) by performing pairwise equijoins instead of equijoin of all the relations in the body of the rule.

Shen et al. [Wei-Min Shen and Zaniolo, 1996] are to best of our knowledge the first who have presented a framework that uses metaqueries to integrate inductive learning methods with deductive database technologies. The confidence measure that we use is similar to the one used by Shen et al. . However, we do not agree with their definition of support, and hence developed our own. According to Shen and Leng [Shen and Leng, 1996], the support (called "base by Shen and Leng) should be the fraction $\frac{|J|}{\Pi_{i=1}^{m}|L_i|}$. We believe that our definition of support reflects better its intuitive meaning. Consider again the students database of Figure 1, and suppose that there are 100,000 tuples in the relation student-course and 1000 tuples in the relation course-department. According to Shen and Leng, the support for the rule (3) will be $\frac{10^5}{10^8} = \frac{1}{1000}$. The support will shrink even more as the number of courses grow, no matter how many students are in the relation student-course. This low support will render the rule (3) uninteresting, although this rule involves all the tuples in the relation student-course. Note that according to our definition, the support for this rule is 1.0, because there is one relation, *student-course* which participate in the equijoin of the body of The rule in full capacity.

2 Complexity

How hard is answering a metaquery? In this section we will show that the computation task is NP-Hard. First, let us rephrase the problem as a decision problem.

Definition 2.1 (The MQ Problem)

Instance: *A database VB and a metaquery MQ of the form(l).*

[1]The equijoins are accomplished by enforcing values of attributes that are bound to the same variable name in the metaquery to be equivalent.

***Question:* Are there relations $q_1, ..., q_m, t$ in $\mathcal{DB}$ such that the rule $T \longleftarrow L_1, ..., L_m$ holds with support and confidence greater than 0 when we replace T by t and each Q_i, $i = 1, ..., m$ by q_i.**

Theorem 2.2 *The MQ problem is NP-Hard.*

Proof: We will show a reduction from HAMILTONIAN PATH (HP) to the MQ problem. We remind the reader that the HP problem is as follows [Garey and Johnson, 1979]: given a directed Graph (V, *E),* determine whether there is a path in the graph by which each vertex is visited exactly once. So suppose we are given a graph *G* = (V, *E).* We will construct a database *VB* and a metaquery *MQ* as follows. *VB* will have two relations: *Rv* and *RE- RV* contains exactly one tuple, which is the list of all vertices in V, and R_E is the set of all edges in *E,* that is:

$$R_V = \{(v_1, v_2, ..., v_n)\}, where V = \{v_1, ..., v_n\}, \text{and}$$
$$R_E = \{(u, v) | (u, v) \in E\}$$

MQ will be the following metaquery:

$$Q_0(X_1, X_2, ..., X_n)$$
$$Q(X_1, X_2), Q(X_2, X_3), ..., Q(X_{n-1}, X_n)$$

We claim that the answer to the MQ problem composed of the above *VB* and *MQ* will be "YES* iff there is a Hamiltonian path $u_1, u_2, ..., u_n$ in *G.* And indeed, if there is a Hamiltonian path in G, then the rule

$$R_V(X_1, X_2, ..., X_n)$$
$$R_E(X_1, X_2), R_E(X_2, X_3), ...R_E(X_{n-1}, X_n)$$

Holds with support > 0 and confidence > 0, and so the answer to the given MQ problem will be "YES". On the other hand, if the answer to this MQ problem is YES, then, since we require that the support and confidence be positive, and since *Rv* has only one tuple and *RE* is the only binary relation, it must be the case that there is a Hamiltonian path in *G.* ■

It is worthwhile to mention a tractable subset. If the database scheme is fixed in advance, and we do not allow a predicate variable to appear more than once in a query, then the number and maximum size of all possible different metaqueries is a constant. In this case every metaquery has a constant number of instantiations and hence can be answered by a join of a constant number of relations.

The reader might wonder whether the complexity analysis done on inductive logic programs (e.g. [Dzeroski *et al,* 1992]) is relevant here. The answer is no. In inductive logic programming the goal is to construct a program that will *generate* a given goal relation out of other relations and positive and negative examples. Here, we are interested in finding to what extent some rule on given relations and goal relation holds. We don't have to find a rule that accurately generates the goal relation, said we do not have negative examples.

```
Instantiate(SR)
Input: SR: a set $R_0, R_1, ..., R_n$ of relation patterns, partially instantiated, each with a set of constraints
       $C_0, ..., C_n$. Initially the constraints sets are all empty.
Output: Relations r0, r1...rn that match the relation patterns in SR with the variables bound to attributes.
  1. If SR is completely instantiated then return SR.
  2. Pick the next uninstantiated relation variable $R_i$ from SR, {here we should use CSP variable
     ordering techniques to choose}
  3. Pick up the next possible instantiation $r$ to $R_i$ ($r$ should meet the constraints in $C_i$ and should
     be of the same arity as $R_i$). {here we should use CSP value ordering techniques to choose}
  4. For each relation pattern $R_j$ not yet instantiated do:
       if $R_j$ has a common field variable $X$ with $R_i$, add to the $C_j$ the constraint that $X$ must be
       bound to an attribute with type $T_X$, where $T_X$ is the type of the attribute that $X$ is
       bound to in $r$.
  5. Call Instantiate recursively.
```

Figure 3: Algotithm for the instantiation stage

3 Efficient Algorithms for Metaqueries

In this section we discuss the algorithms for generating all rules resulting from a given metaquery.

The process of answering a metaquery can be divided into two stages. In the first stage, which we call the *instantiation stage,* we are looking for sets of relations that match the pattern determined by the metaquery. In the second stage, which we call the *filtration stage,* we filter out all rules that match the pattern of the metaquery but do not have enough support and confidence.

3.1 The instantiation stage

The process of instantiating a metaquery is similar to solving a Constraint Satisfaction problem (CSP) [Dechter, 1992] where we sure basically looking for all solutions of the CSP problem. In our experiments we used a very simple CSP algorithm (forward checking with Back-jumping [Dechter, 1990]) but other more advanced algorithms may be used.

The basic instantiation algorithm is shown in Figure 3. If this stage suceeds, at the end of this stage each relation pattern *$R(X_1,...,X_n)$* that appear in the metaquery is instantiated. That is, *R* is bound to some relation name r and each variable is bound to an attribute ("field") of the relation. We assume that a procedure *att(r, X)* can return the attribute in r to which the variable *X* is bound.

3.2 The filtration stage

The filtration stage itself is composed of two steps: filtering out rules with low support, and filtering out rules

```
compute-support($r_1, ..., r_m$)
Input: Set of relations $r_1, ..., r_m$, where each of the attributes is bound to a variable. A support threshold $MinSupport$.
Output: if the rule who's body is $r_1, ..., r_m$ has support equal or larger than $MinSupport$, return the support, otherwise return $-1$.
1. $RelSetCopy$ = $RelSet$ = $\{r_1, ..., r_m\}$; $LowSupp$ = true;
2. While ($RelSet \neq \emptyset$) and $LowSupp$ do
    1. Let $r \in RelSet$;
    2. $s$ = $S_i$-upbound($r$, $RelSetCopy$);
    3. If $s \geq MinSupport$ then $LowSupp$ = false
       else $RelSet = RelSet - \{r\}$;
3. If $LowSupp$ then return $-1$
   else return Join-support($r_1, ..., r_m$)
```

Figure 4: computing support for a rule body

with low confidence. We compute confidence only for rules with sufficient support. In our research we have focused so far on algorithms for computing support. We will discuss three alternatives:

The Join approach the straightforward way: computing the equijoin of the body of the rule, then computing S_i (S_i as defined in section 1.1) for each relation in the body, and then taking the maximum.

The histogram approach Using histograms for estimating support. Computing the support using the Join approach only for rules with high estimated support.

The histogram + memory approach same as the histogram approach, except that we store intermediate results in memory, and reuse them when we are called to make the same computation.

The procedure **Join-support($r_1, ..., r_m$)** computes the support S_i of each relation by performing the Join as explained in section 1.1. and then returns $\mathbf{Max}\{S_i | 1 \leq i \leq m\}$. Since the Join is an expensive operation, we try to detect rules with low support using some low-cost procedures. The other two approaches compute an upper bound on the support and then compute the exact support only for rules with high enough upper bound of support. The idea is summarized in Algorithm compute-support in Figure 4. Note that once one relation with high S_i is found, we turn to compute the exact support using Join.

The procedure S_i-upbound called by the algorithm compute-support returns an upper bound for the value S_i for a single relation r_i in the body of the rule. This can be done by one of the two procedure: S_i-upbound-brave or S_i-upbound-cautious, shown in Figures 5 and 6, respectively. The basic idea is that an upper bound can be achieved by taking the join of a relation ri with any other

```
$S_i$-upboud-brave($r_i$,$S$)
input: A relation $r_i$ and a set of relations $S$.
output: An upper bound on $S_i$ for a rule whose body is $S \bigcup \{r_i\}$.
1. $s = 1.0$
2. If there is $r' \in S$ such that $r'$ and $r_i$ have a common variable $X$
   then $s$ =upbound($r_i$, $r'$, $att(r_i, X)$, $att(r', X)$);
3. return $s$;
```

Figure 5: computing S_i bravely

```
$S_i$-upboud-cautious($r_i$,$S$)
input: A relation $r_i$ and a set of relations $S$.
output: An upper bound on $S_i$ for a rule whose body is $S \bigcup \{r_i\}$.
1. $s = 1.0$
2. for each relation $r'$ in $S$ such that $r_i$ and $r'$ have variables in common
       do for each common variable $X$ of $r_i$ and $r'$
          do
             $s'$ =upbound($r_i$, $r'$, $att(r_i, X)$, $att(r', X)$);
             if $s' < s$ then $s$=$s'$;
3. return $s$;
```

Figure 6: computing S_i cautiously

relation with which r_i has a variables in common. Procedure S_i-upbound-brave does this by picking one arbitrary relation with which r_i has a common variable, and procedure S_i-upbound-cautious does this by considering *all* relations with which r_i has variables in common, and taking the minimum. Procedure S_i-upbound-cautious works harder than procedure S_i-upbound-brave but it achieves a tighter upper bound and hence can save more Join computations.

The procedure *upbound* called by procedure S_i-upbound-cautious or procedure S_i-upbound-brave founds an upper bound on S_i (as in Section 1.1) for a given relation r_i using one of two approaches: the histogram approach or the histogram+memory approach.

The histograms approach exploits the fact that histograms are easy to construct and are quite useful for support estimation. A histogram of an attribute of some relation is a mapping *h* between the set of values that this attribute can take and the set of natural numbers, such that for each possible value v, *h(v)* is the number of tuples in the relation in which *v* appears as the value of the attribute. Given two relations r_1 and r_2, procedure upbound-histo shown in Figure 7 achieves an upper bound on the support of the equijoin between them. Procedure upbound-histo prepares the histograms and then calls the procedure Histo which actually computes the

```
upbound-histo($r_1, r_2, att_1, att_2$)
input: Two relations $r_1$ and $r_2$, $att_1$ an attribute of $r_1$,
    $att_2$ an attribute of $r_2$. $att_1$ and $att_2$ are of the same
    type.
output: An upper bound on the support where only $r_1$
    and $r_2$ are considered.
1. Let $U$ be the set of all distinct values in $att_1$ of $r_1$ and
   $att_2$ of $r_2$.
2. Let $h_1$ be a histogram with domain $U$ of the values in
   $att_1$ of $r_1$. (If there is no such histogram, build it).
3. Let $h_2$ be a histogram with domain $U$ of the values in
   $att_2$ of $r_2$. (If there is no such histogram, build it).
4. return Histo($r_1, r_2, h_1, h_2, |U|$);
```

Figure 7: Computing support from histograms

```
Histo($r_1$,$r_2$,$h_1$,$h_2$,$n$)
input: Two relations $r_1$, $r_2$, and two histograms $h_1, h_2$
    reflecting how many times a value of some common
    attribute appears in each of them, respectively. We
    assume that there are $n$ distinct possible values.
output: An upper bound on the support where only $r_1$
    and $r_2$ are considered.
1. set support1 = 0; support2 = 0
2. for i =1 to n
     if $h_1$(i)> and $h_2$(i)> 0 then
         1. support1 = support1 + $h_1$(i)
         2. support2 = support2 + $h_2$(i)
3. return max(support1/size($r_1$), support2/size($r_2$) )
```

Figure 8: The procedure Histo

support.

According to Claim 1.1 of Section 1.1, the number returned by Histo is also an upper bound on the support of the rule in which only r_1 and r_2 appear in the body with a common variable *X.*

The problem with the procedure upbound-histo is that it doesn't exploit the fact that pairs of instantiated relations can appear again and again in many different instantiations of the same metaquery. In the procedure upbound-histo-mem, shown in Figure 9, we save in memory estimated support of pairs of relations and retrieve this information if necessary.

3.3 Evaluation

The efficiency of algorithm compute-support depends on the likelihood of finding a rule with high support. If a large fraction of the rules has a high support, then this algorithm will work harder then the straightforward algorithm which computes support by performing Join without trying to estimate the result before. Note that for rules with high support algorithm compute-support works harder than the algorithm which performs a Join

```
upbound-histo-mem($r_1, r_2, att_1, att_2$)
input: Two relations $r_1$ and $r_2$, $att_1$ an attribute of $r_1$,
    $att_2$ an attribute of $r_2$. $att_1$ and $att_2$ are of the same
    type.
output: An upper bound on the support where only $r_1$
    and $r_2$ are considered.
note: This procedure uses a procedure fetch-supp-
    mem($r_1, r_2, att_1, att_2$) which returns the estimated
    support for these two relations on these attributes as
    recorded in memory. If no such estimate is recorded,
    it returns $-1$. Similarly, the procedure put-supp-
    mem($s, r_1, r_2, att_1, att_2$) stores the estimated support
    $s$ for these two relations on these attributes in mem-
    ory.
1. $s$ = fetch-supp-mem($r_1, r_2, att_1, att_2$)
2. If $s = -1$ then
       1. $s$ = upbound-histo($r_1, r_2, att_1, att_2$);
       2. put-supp-mem($s, r_1, r_2, att_1, att_2$);
3. return $s$;
```

Figure 9: Computing support from histograms and memory

directly because first it estimates the support, finds out that it is high, and then calls the Join procedure.

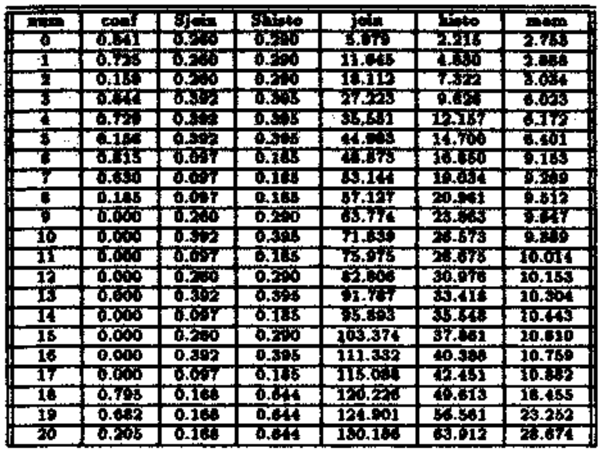
Our working assumption is that rules with high support are much less likely. In any case, however, the above analysis calls for an experimental evaluation of the algorithms. In this subsection we present some preliminary results on such experiments. The evaluation was done on the Studentgrades database supported by the FlexiMine system [Domshlak *et* a/., 1998]. This database contains information on students and some of their demographic characteristics, courses, and grades. The relevant tables for our experiment are: The *student* table with 997 rows, The *course* table with 1403 rows, The *family* table with 997 rows, The *studentcourse* table with 20705 rows. Each table contains between 5 and 9 attributes.

We have compared the performance of our algorithms by measuring the time it took them to compute support of 20 different rules involving four relations each. All the rules were instances of the same metaquery. The experiments were done on a Sun / SunOS workstation with one spare CPU and 128 MB main memory.

The time (in seconds) for the support computation were measured for the following configurations:

1. the support is computed by performing the join (procedure Join-support),
2. the support is computed by histogram without using memory (procedure upbound-histo in Figure 7).
3. the support is computed by our histogram using memory (procedure upbound-histo-mem in Figure 9)-

All the above methods were tested using the procedure 5,-upbound-brave. Procedure S_t-upbound-cautious was not tested yet.



num	conf	Sjoin	Shisto	join	histo	mem
0	0.541	0.260	0.290	5.979	2.216	2.753
1	0.725	0.260	0.290	11.645	4.830	2.858
2	0.159	0.260	0.290	18.112	7.322	3.034
3	0.644	0.392	0.395	27.223	9.626	6.023
4	0.729	0.392	0.395	35.551	12.157	6.172
5	0.166	0.392	0.395	44.903	14.700	6.401
6	0.815	0.097	0.185	48.875	16.650	9.153
7	0.630	0.097	0.185	53.144	18.634	9.389
8	0.185	0.097	0.185	57.127	20.961	9.512
9	0.000	0.260	0.290	63.774	23.863	9.647
10	0.000	0.392	0.395	71.839	26.573	9.889
11	0.000	0.097	0.185	75.975	28.675	10.014
12	0.000	0.260	0.290	82.806	30.976	10.153
13	0.600	0.392	0.395	91.787	33.418	10.304
14	0.000	0.097	0.185	95.893	35.648	10.443
15	0.000	0.260	0.290	103.374	37.861	10.610
16	0.000	0.392	0.395	111.332	40.386	10.759
17	0.000	0.097	0.185	115.084	42.451	10.882
18	0.795	0.168	0.644	120.226	49.613	18.455
19	0.682	0.168	0.644	124.901	56.561	23.252
20	0.205	0.168	0.644	130.186	63.912	28.674

Figure 10: Comparison of support computations

Table 10 shows the results obtained when the support threshold was set to 0.5. The columns is as follows:

num - the serial number of the rule, conf- the confidence of the rule.

Sjoin - the support computed by the definition, i.e first performing the join and then computing the support

Shisto - the estimated support computed by our Histo method (Figure 8).

join - the time to compute the support by the Join method, *accumulated.*

histo - the time to compute the support by the histogram method, *accumulated,* including time for computing the histograms.

mem - the time to compute the support by the Histogram method with memory, *accumulated.*

It can be seen that the Histogram method achieves about 50% savings in time, while the memory method has even improved on that.

Figure 11 shows the behavior of all three methods with a varying support threshold (0 — 0.5). Line 1 is the time for the Join method, Line 2 - for the histogram method, and Line 3 for the histogram+memory method. In that figure the time for the Join method is normalized to 1, while the other time lines are relative to this time (e.g. the value 0.8 for the histogram method means that for that value of support, the histogram method time was 0.8 of the Join method). It can be seen that the larger is the support threshold, the better the methods which estimate first perform. This is because the highest the threshold is, the more rules do not pass it and can be cut by using support estimates.

4 Conclusion

This paper contributes to the research on metaqueries in several ways. We analyze the complexity of the related computational problem, we propose a new notion of support for a rule generated according to a pattern, and we present novel and efficient algorithms for computing

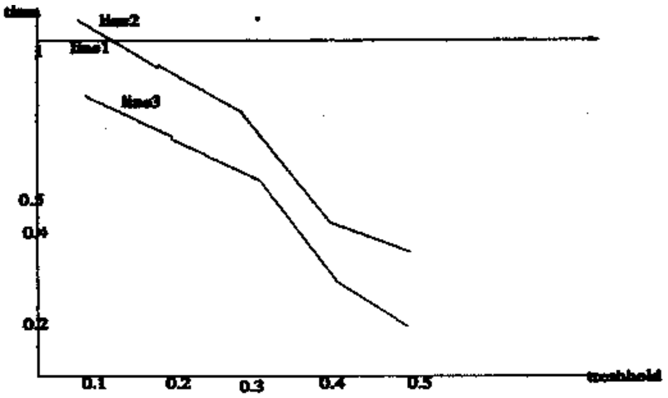


Figure 11: Comparison of support computations

support. Although more experimental work is needed for real evaluation of the algorithms we have developed, preliminary experimental evaluation is quite promising.

References

[Dechter, 1990] Rina Dechter. Enhancement schemes for constraint processing: Backjnmping, learning, and cutset decomposition. 41:273-312, 1990.

[Dechter, 1992] Rina Dechter. Constraint networks. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence,* pages 276-285. John Wiley, 2nd edition, 1992.

[Domshlak *et* of., 1998] C. Domshlak, D. Gershkovich, E. Gndes, N. Liusternik, A. Meiseb, T. Rosen, and S. E. Shimony. FlexiMine - A Flexible Platform for KDD Research and Application Construction. Technical Report version FC-98-04, Ben-Gurion University, 1998. KDD-98, Fourth International Conference on Knowledge Discovery in Databases.

[Dzeroslri *et al.,* 1992] S. Dzeroslri, S. Muggleton, and S. Russell. Pac-learnability of determinate logic programs. In *Proceedings of the Fifth ACM Workshop on Computational Learning Theory,* pages 128-135, New York, 1992.

[Fayyad *et* al. , 1996] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI Press/MIT Press, 1996.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and intractability, A guide to the theory of NP-completeness.* W. H. Freeman and Company, 1979.

[Shen and Leng, 1996] Wei-Min Shen and Bing Leng. A metapattern-based automated discovery loop for integrated data mining- unsupervised learning of relational patterns. *IEEE Transactions on Knowledge and Data Engineering,* 8(6):898-910, 1996.

[Shen, 1992] W. M. Shen. Discovering regularities from knowledge bases. *Intelligent Systems,* 7(7):623-636, 1992.

[Wei-Min Shen and Zaniolo, 1996] B. Mitbander Wei-Min Shen, K. Ong and C. Zaniolo. Metaqueries for Data Mining. In *Advances in Knowledge Discovery and Data Mining,* pages 375-397. AAAI/MIT press, 1996.