International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 7, September - 2012

Identifying Peculiarities to Exploit Perceptibility of Widgets

M. Vijaya Lakshmi[1], D. Uma Devi[2]

[1]*M.Tech II year Student, Department of CSE, Sri Mittapalli College of Engineering,*

[2]*Associate Professor, Department of CSE, Sri Mittapalli College of Engineering,*

Abstract

In recent years, there has been important concentration in the expansion of ranking functions and capable top-k rescue algorithms to help users in ad-hoc search and rescue in databases (e.g., buyers searching for products in a catalog). We introduce a complementary problem: how to guide a seller in picking the best peculiarities of a new tuple (e.g., a new product) to highlight so that it stands out in the crowd of existing competitive products and is widely visible to the pool of potential buyers. We develop several formulations of this problem. Although the problems are NP-complete, we give several exact and approximation algorithms that work well in practice. One type of exact algorithms is based on Integer Programming (IP) formulations of the problems. Another class of exact methods is based on maximal regular itemset mining algorithms. The approximation algorithms are based on greedy heuristics. A detailed performance study illustrates the benefits of our methods on real and synthetic data.

Key Words: *Data mining, information and facts, engineering tools and procedures, advertising, withdrawal methods and algorithms, rescue designs.*

I. INTRODUCTION

In recent years, there has been significant interest in developing effective procedures for ad-hoc search and rescue in unstructured as well as structured data repositories, such as text collections and relational databases. In particular, a large number of emerging applications require exploratory querying on such databases; examples include users wishing to search databases and catalogs of products such as mobiles, cars, cameras, restaurants, or articles such as news and job ads. Users browsing these databases typically execute search queries via public front-end interfaces to these databases. Typical queries may specify sets of keywords in case of text databases, or the desired values of certain peculiarities in case of structured relational databases. The query-answering system answers such queries by either returning all data objects that satisfy the query conditions, or may rank and return the *top-k* data objects, or return the results that are on the query's skyline. If ranking is employed, the ranking may either be simplistic – e.g., objects are ranked by an attribute such as Price; or more sophisticated – e.g., objects may be ranked by the degree of *"relevance"* to the query. While unranked rescue (also known as *Boolean Rescue*) is more common in traditional SQL-based database systems, ranked rescue (also known as *Top-k Rescue*) is more common in text databases, e.g. tf-idf ranking [20]. Recently there has been widespread interest in developing suitable *top-k* rescue procedures even for structured databases [1, 7, and 30]. Skyline rescue semantics is also investigated where a data point is retrieved by a query if it is not dominated by any other data point in all dimensions [4, 19, 22, 25, 29, and 31].

Screening Peculiarities for greatest Perceptibility: We distinguish between two types of users of these databases: users who search such databases trying to locate objects of interest, and users who insert new objects into these databases in the hope that they will be easily discovered by the first type of users. For example, in a database representing an e-marketplace the former type of users are *potential buyers* of products; while the latter type of users are *sellers of products*. Almost all of the prior research efforts on effective search and rescue procedures – such as new top-k algorithms, new relevance measures, and so on – have been designed with the first kind of user in mind (i.e., the buyer). In contrast, less research has been addressing procedures to help a seller/manufacturer insert a new product for sale in such databases that markets it in the best possible manner – i.e., such that it stands out in a crowd of competitive products and is widely visible to the pool of potential buyers.

It is this latter problem that is the main focus of this paper. To understand it a little better, consider the following scenario: assume that we wish to insert a classified ad in an online newspaper to advertise about cars, mobiles and cameras for sale. Our products may have numerous peculiarities (it will have power window, four door, petrol, diesel, etc). However, due to the ad costs involved, it is not possible for us to describe all peculiarities in the ad. So we have to select, say the ten best peculiarities. Which ones should we select? Thus, one may view our effort as an attempt to build a recommendation system for *sellers*, unlike the more traditional recommendation systems for buyers. It may also be viewed as *inverting* a ranking function, i.e., identifying the argument of a ranking function that will lead to high ranking scores.

This general problem also arises in domains beyond ecommerce applications. For example, in the design of a new product, a manufacturer may be interested in picking the ten best features from a large wish-list of possible features – e.g., new car can have the feature of high mileage. To define our problem more formally, we need to develop a few abstractions. Let D be the database of products already being advertised in the marketplace (i.e., the "competition"). In addition, let Q be the set of search queries that have been executed against this database in the recent past – thus Q is the "workload" or "query register". For a new product that needs to be inserted into this database, we assume that the seller has a complete "ideal" description of the product. Our problem can now be defined as follows:

PROBLEM:

Given a database D, a query register Q, a new tuple t, and an integer m, determine the best (i.e., top- m) peculiarities of t to retain such that if the shortened version of t is inserted into the database, the number of queries of Q that retrieve t is exploited.

We consider several variants, including Boolean, categorical, text and numeric data, and conjunctive and disjunctive query semantics. We also consider variants in which the "budget", i.e., m, is not specified; in this case our objective is to determine the value of m such that the number of satisfied queries divided by m is exploited. Thus we seek to exploit the "per dollar" benefit. A special case of this no- budget variant is when ranking is performed using functions that are non-monotone on the number of specified peculiarities (keywords), such as the BM25 [18] scoring function used in Information Rescue.

We analyze the computational complexity of these problems, and show that most variants are NP-complete. We develop two types of methods yielding optimal solutions: (a) procedures based on Integer Programming (IP) and Integer Linear Programming (ILP) methods, which work well for moderate-sized problem instances, and (b) more scalable solutions based on novel adaptations of maximal regular set algorithms that also allow us to leverage several preprocessing opportunities.

Main Contributions:

The main contributions of this paper may be summarized as follows:

1. We introduce the problem of *identifying peculiarities of a tuple for exploit perceptibility* as a new data exploration problem. We consider several interesting variants of the problem as well as diverse application scenarios.
2. We analyze the computational complexity of the different variants of the problem and show that most of them are NP-complete.
3. We develop optimal Integer Programming (IP) and Integer Linear Programming (ILP) based algorithms to solve certain variants of the problem. These algorithms are effective for moderate-sized problem instances.
4. For certain problem variants, we also develop more scalable optimal solutions based on novel adaptations of maximal regular itemset algorithms. Furthermore, in contrast to ILP-based solutions, we can leverage preprocessing opportunities in these approaches.
5. We also develop fast greedy approximation algorithms that work well for all problem variants.
6. We perform detailed performance evaluations on both real as well as synthetic data to demonstrate the effectiveness of our developed algorithms.

II. FUNDAMENTALS

First we provide some useful definitions.

Boolean Database: Let $D = \{t_1 \ldots t_N\}$ be a collection of Boolean tuples over the attribute set $A = \{a_1 \ldots a_M\}$, where each tuple t is a bit-vector where a_0 implies the absence of a feature and a_1 implies the presence of a feature. A tuple t may also be considered as a subset of A, where an attribute belongs to t if its value in the bit-vector is 1.

Tuple Domination: Let t_1 and t_2 be two tuples such that for all peculiarities for which tuple t_1 has value 1, tuple t_2 also has value 1. In this case we say that t_2 **dominates** t_1.

Tuple Compression: Let t be a tuple and let t' be a subset of t with m peculiarities. Thus t' represents a compressed representation of t. equally, in the bit-vector representation of t, we retain only m 1's and convert the rest to 0's.

Query Register: Let $Q = \{q_1 \ldots q_S\}$ be collection of queries where each query q defines a subset of peculiarities.

III. MAJOR TROUBLES DEVIATION: CONJUNCTIVE BOOLEAN WITH QUERY REGISTER

In this section we formally define the main problem for Boolean data. In Section 3.1 we formally define the problem and in Section 4 we provide algorithms for this problem variant.

Conjunctive Boolean Rescue: We view each query as a conjunctive query. A tuple t satisfies a query q if q is a subset of t. For example, a query such as $\{a_1, a_3\}$ is equivalent to "return all tuples such that $a_1 = 1$ and $a_3 = 1$". Alternatively, if we view q as a special type of "tuple", then t **dominates** q. The set of returned tuples R (q) is the set of all tuples that satisfy q. In this problem variant as well as in most of the variants defined later, our task is to compress a new tuple t by retaining the best set of m peculiarities (i.e., top-m peculiarities) such that some criterion is optimized.

3.1 Problem Definition

Conjunctive Boolean - Query Register (CB-QR): *Given a query register Q with Conjunctive Boolean Rescue semantics, a new tuple t, and an integer m, computes a compressed tuple t′ having m peculiarities such that the number of queries that retrieve t′ is exploited.*

Intuitively, for buyers interested in browsing products of interest, we wish to ensure that the compressed version of the new product is visible to as many buyers as possible. This problem also has a *per-attribute* version where *m* is not specified; in this case we may wish to determine *t′* such that the number of satisfied queries divided by **|*t′*|** is exploited. Intuitively, if the number of peculiarities retained is a measure of the cost of advertising the new product, this problem seeks to exploit the number of potential buyers per advertising dollar.

CAM ID	Flash	Pixel	Battery	TV Kit	Memory
t_1	0	1	0	1	0
t_2	0	1	1	0	0
t_3	1	0	0	1	1
t_4	1	1	0	1	0
t_5	1	1	0	0	0
t_6	0	1	0	1	0
t_7	0	0	1	1	0

DATABASE D

Query ID	Flash	Pixel	Battery	TV Kit	Memory
q_1	1	0	1	1	1
q_2	1	1	0	0	0
q_3	0	1	1	0	0
q_4	0	0	1	0	1
q_5	1	1	0	1	0

Query Register Q

New CAM	Flash	Pixel	Battery	TV Kit	Memory
t	1	1	0	0	1

New tuple t to be inserted

Fig 1: Illustrating the Fundamentals

IV. ALGORITHMS FOR CONJUNCTIVE BOOLEAN WITH QUERY REGISTER

In this section we discuss our main algorithmic results for the main problem variant discussed in Section 3.

4.1 Optimal Brute Force Algorithm

Clearly, since CB-QR is NP-hard, it is unlikely that any optimal algorithm will run in polynomial time in the worst case. The problem can be obviously solved by a simple brute force algorithm (henceforth called *Brute- Force-CB-QR*), which simply considers all combinations of *m*-peculiarities of the new tuple *t* and determines the combination that will satisfy the maximum number of queries in the query register Q. However, we are interested in developing optimal algorithms that work much better for typical problem instances.

4.2 Optimal Algorithm based on Maximal Regular Itemsets

The algorithm based on Integer Linear Programming described in the previous subsection has certain limitations; it is impractical for problem instances beyond a few hundred queries in the query register. The reason is that it is a very generic method for solving arbitrary integer linear programming formulations, and consequently fails to leverage the specific nature of our problem. In this subsection we develop an alternate approach that scales very well to large query registers. This algorithm, called Max-FreqItemSets-CB-QR, is based on an interesting adaptation of an algorithm for mining Maximal Regular Itemsets [7].

4.2.1 The Regular Itemset Problem

Let R be an N-row Mcolumn Boolean table, and let r > 0 be an integer known as the verge. Given an itemset I (i.e., a subset of peculiarities), let freq (I) be defined as the number of rows in R that "support" I (i.e., the set of peculiarities corresponding to the 1's in the row is a superset of I). Compute all itemsets I such that freq (I) > r.

Computing regular itemsets is a well studied problem and there are several scalable algorithms that work well when R is sparse and the verge is suitably large. Examples of such algorithms include [2, 15]. In our case, given a new tuple t, recall that our task is to compute t′, a compression of t by retaining only m peculiarities, such that the number of queries that satisfy t′ is exploited. This immediately suggests that we may be able to leverage algorithms for regular itemsets mining over Q for this purpose. However, there are several important complications that need to be overcome, which we elaborate next.

4.2.2 Balancing the Query Register

Firstly, in itemset mining, a row of the Boolean table is said to support an itemset if the row is a superset of the itemset. In our case, a query satisfies a tuple if it is a subset of the tuple. To overcome this conflict, our first task is to complement our problem instance, i.e., convert 1's to 0's and vice versa. Let ~t (~q) denote the complement of a tuple t (query q), i.e., where the 1's and 0's have been interchanged. Likewise let ~Q denote the complement of a query register Q where each query has been complemented. Now, freq (~t) can be defined as the number of rows in ~Q that support ~t. Rest of the approach is now seemingly clear: compute all regular itemsets of ~Q (using an

appropriate verge to be discussed later), and from among all regular itemsets of size M – m, determine the itemset I that is a superset of ~t with the highest frequency. The optimal compressed tuple t' is therefore the complement of I, i.e., ~I. However, the problem is that Q is itself a sparse table, as the queries in most search applications involve the specification of just a few peculiarities. Consequently, the complement ~Q is an extremely dense table and this prevents most regular itemset algorithms from being directly applicable to ~Q. For example, most "level-wise algorithms" (such as Apriori [1], which operates level by level of the Boolean lattice over the peculiarities set by first computing the single itemsets, then itemsets of size 2, and so on) will only progress past just a few initial levels before being overcome by an intractable explosion in the size of candidate sets. To see this, consider a table with M=50 peculiarities, and let m = 10. To determine a compressed tuple t′ with 10 peculiarities, we need to know the itemset of ~Q of size 40 with maximum frequency. Due to the dense nature of ~Q, algorithms such as Apriori will not be able to compute regular itemsets beyond a size of 5-10 at the most. Likewise, the sheer number of regular itemsets will also prevent other algorithms such as FP-Tree [9] from being effective.

We have developed an adaptation of regular itemset mining algorithms to overcome this problem of extremely dense datasets. Before we describe details of our approach, let us discuss the issue of how the verge parameter should be set.

4.2.3 Setting of the Verge Parameter

We can solve the problem of itemset mining of extremely dense datasets. What should be the setting of the verge? Clearly setting the verge r=1 will solve CB-QR optimally. But this is likely to make any itemset mining algorithm impractically slow.

There are two alternate approaches to setting the verge. One approach is essentially a heuristic, where we set the verge to a reasonable fixed value dictated by the practicalities of the application. The intuition is that the verge enforces that peculiarities should be selected such that the compressed tuple is satisfied by a certain minimum number of queries. For example, a verge of 1% means that we are not interested in results that satisfy less than 1% of the queries in the query register, i.e., we are attempting to compress t such that at least 1% of the queries are still able to retrieve the tuple. It is important to note that for a fixed verge setting such as this, one of two possible outcomes can occur. If the optimal compression t′ satisfies more than 1% of the queries, the algorithm will discover it. If the optimal compression satisfies less than 1% of the queries, then the algorithm will return empty.

We also suggest an alternate adaptive procedure of setting the verge that is guaranteed to find the optimal compression. First initialize the verge to a high value and compute the regular itemsets of ~Q. If there are no regular itemsets of size at least M – m that are supersets of ~t, repeat the process with a smaller verge which is half of the previous verge.

We now return to the task of how to compute regular itemsets of the dense Boolean table ~Q. In fact, we do not compute all regular itemsets of the dense table ~Q, as we have already argued earlier that there will be prohibitively too many of them. Instead, our approach is to compute the *maximal regular itemsets* of ~Q.

4.2.4 Random Walk to Compute Maximal Regular

Itemsets: A maximal *regular itemset* is a regular itemset such that none of its supersets are regular. The set of maximal regular itemsets are much smaller than the set of all regular itemsets. For example, if we have a dense table with M peculiarities, then it is quite likely that most of the maximal regular itemsets will exist very high up in the Boolean lattice over the peculiarities, very close to the highest possible level M. Fig 2 shows a conceptual diagram of a Boolean lattice over a dense Boolean table ~Q. The shaded region depicts the regular itemsets and the maximal regular itemsets are located at the highest positions of the border between the regular and irregular itemsets.

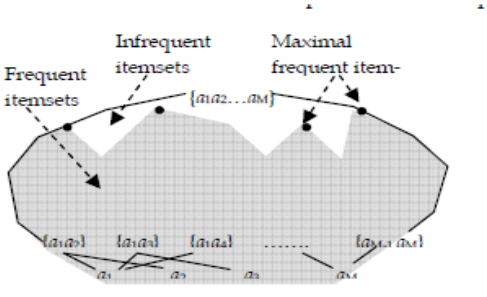

Fig 2: Maximal Regular Itemsets in a Boolean Lattice

There exist several algorithms for computing maximal regular itemsets, e.g. [3, 5, 12, and 14]. We base our approach on the random walk based algorithm in [7], which starts from a random singleton itemset I at the bottom of the lattice, and at each iteration, adds a random item to I (from among all items A - I such that I remains regular), until no further additions are possible. At this point a maximal regular itemset I have been discovered. If the number of maximal regular itemsets is relatively small, this is a practical algorithm: repeating this random walk a reasonable number of times will with high probability discover all maximal regular itemsets. However, since this algorithm is based on traversing the lattice from bottom to top, it implies that the random walk will have to traverse a lot of levels before it reaches a maximal regular itemset of a dense table.

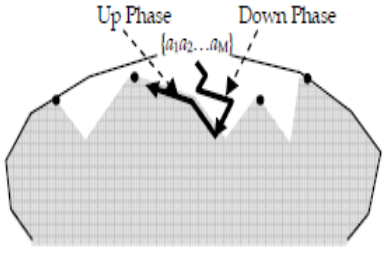

Fig 3: Two Phase Random Walk

Instead, we propose an alternate approach which starts from the top of the lattice and traverses down. Our random walk can be divided into two phases: (a) Down Phase: starting from the top of the lattice (I = $\{a_1a_2\ldots a_M\}$), walk down the lattice by removing random items from I until I becomes regular, and (b) Up Phase: starting from I, walk up the lattice by adding random items to I (from among all items A - I such that I remains regular), until the no further additions are possible. At this point a maximal regular itemset I have been discovered.

Fig 3 shows an example of the two phases of the random walk. What is important to note is that this process is much more capable than a bottom-up traversal, as our walks are always confined to the top region of the lattice and we never have to traverse too many levels. Complementing the query register eventually results in a dense dataset. In a dense dataset, maximal regular itemsets are usually at the top of the lattice. That is, they are close to level M - m, where m is comparatively much smaller than M. If a bottom-up approach is used to find maximal regular itemsets, it will have to traverse a long portion of the lattice (i.e., too many levels) and will be incapable. Whereas, in top-down approach, the first phase tries to find the first regular itemset along the path from the top which is usually close to a maximal regular itemset, the walks are confined to the top region of the lattice and we never have to traverse too many levels.

4.2.5 Complexity Analysis of a Random Walk Sequence

In the worst case, the cost of a down-up random walk is 2.M.|Q|, where M is the total number of peculiarities and |Q| is the size of the query register. Although in the worst case the random walk will go up and down the whole lattice, in practice we only expect each portion of the walk to traverse only a few levels at the top of the lattice.

4.2.6 Number of Iterations

Repeating this two phase random walk several times will discover, with high probability, all the maximal regular itemsets. The actual number of such iterations can be monitored adaptively; our approach is to stop the algorithm if each discovered maximal regular itemset has been discovered at least twice (or a maximum number of iterations have been reached). This stopping heuristic is motivated by the Good-Turing estimate for computing the number of different objects via sampling [4].

4.2.7 Regular Itemsets at Level M – *m*

Finally, once all maximal regular itemsets have been computed, we have to check which ones are supersets of ~t. Then, for all possible subsets (of size M – m) of each such maximal regular itemset (see Fig 4), we can determine that subset I that is (a) a superset of ~t, and (b) has the highest frequency.

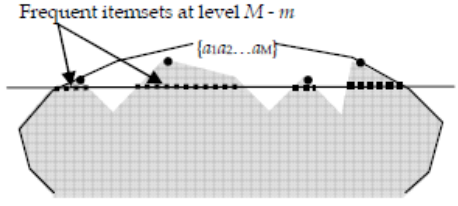

Fig 4: Checking Regular Itemsets at Level M-*m*

The optimal compressed tuple t′ is therefore the complement of I, i.e., ~I. In summary, the pseudo-code of our algorithm Max- FreqItemSets-CB-QR is shown in Fig. 5. Details of how certain parameters such as the verge are set, are omitted from the pseudo-code.

4.2.8 Preprocessing Opportunities

Note that the algorithm also allows for certain operations to be performed in a preprocessing step. For example, all the maximal itemsets can be recomputed, and the only task that needs to be done at runtime is to determine, for a new tuple t, those itemsets that are supersets of ~t and have size M – m. If we know the range of m that is usually requested for compression in new tuples, we can even recomputed all regular itemsets for those values of m, and lookup the itemset with the highest frequency at runtime.

4.2.9 The Per-Attribute Variant

CB-QR has a per attribute variant, where m is not provided as an input, and we have to determine the best m such that number of satisfied queries divided by m is exploited. This variant can be simply solved by trying out values of m between 1 and M and making M calls to any of the algorithms discussed above, and picking the solution that exploits our objective. Since we adopt this general strategy for all per-attribute problem variants, we do not discuss such variants any further in this paper.

```
Q: Query Log
t: new tuple
m: num attributes of t to retain
r: threshold ← suitable value
MaxFreqItemsets ← {}
MaxNumIter ← suitable value
Algorithm TwoPhase-Random-Walk(~Q, r)
  execute Down Phase random walk
  execute Up Phase random walk
  return itemset reached after Up Phase
Algorithm ComputeMaxFreqItemsets(~Q, r)
  while
   (i++ ≤ MaxNumIter) and
   (∃ J in MaxFreqItemsets s.t.
      timesDiscovered(J)= 1)
   I ← TwoPhase-Random-Walk(~Q, r)
   timesDiscovered(I)++
   MaxFreqItemsets←MaxFreqItemsets ∪ {I}
Algorithm MaxFreqItemSets-CB-QL(~Q, r)
  ComputeMaxFreqItemsets(~Q, r)
  let Itemsets(t) ← {I | I ⊆ MaxFreqItemsets,
                        |I| = M – m, and I ⊇ ~t)}
  let I be the itemset in Itemsets(t) with highest
     frequency
  return ~I
```

Fig 5: Algorithm *MaxFreqItemSets*-CB-QR

V. PROBLEMS VARIANT FOR TEXT DATA

5.1 Text Data Problem Definition

A text database consists of a collection of documents, where each document is modeled as a bag of words as is common in Information Rescue. Queries are sets of keywords, with top-k rescue via query-specific scoring functions, such as the tf-idf-based BM25 scoring function [18]. This problem arises in several applications, e.g., when we wish to post a classified ad in an online newspaper and need to specify important keywords that will enable the ad to be visible to the maximum number of potential buyers. A subtle point is that, due to the non-monotonicity of many IR ranking functions, it is possible that a top-m1 tuple compression is worse (fewer queries retrieve document t') than a top-m_2 compression, where $m_1 > m_2$. The attribute selection problem for text data is also NP-complete as it can be converted into Boolean problem considering each keyword as a Boolean attribute. The problem is NP-hard for the case of monotone ranking function, as in CB-QR. Hence, it is also NP-hard for the more complex non-monotonic ranking functions.

5.2. Algorithms for Text Data

As discussed above, text data can be treated as Boolean data, and all the algorithms developed for Boolean data can be used for text data. There are two issues that we wish to highlight, however. One is that if we view each distinct keyword in the text corpus (or query register) as a distinct Boolean attribute, the dimension of the Boolean database is enormous. Consequently, none of the optimal algorithms, either IP-based or regular itemset-based, are feasible for text data. Fortunately, the greedy heuristics we have developed scale very well with reasonable results, as described in the experiments section. The second issue is that some of the scoring function that are used in text data – e.g., the BM25 scoring function that takes into account the document length (size of compressed tuple t′) – are non-monotonic on the number of keywords added. In particular, adding a query keyword to t′ may decrease its BM25 score if this keyword has very low inverse document frequency (idf). Consequently the per-attribute versions of our various problem variants are of interest.

VI. EXPERIMENTS

In this section we describe the experimental setting and the results. Our main performance indicators are (a) the time cost of the proposed optimal and greedy algorithms, and (b) the approximation quality of the greedy algorithms, for the CB-QR and text data problem variants presented in Sections 3 and 5 respectively.

System Configuration: We used Microsoft SQL Server 2000 RDBMS on a P4 3.2-GHZ PC with 1 GB of RAM and 100 GB HDD for our experiments. Algorithms are implemented in C#, and connected to RDBMS through ADO.

Datasets: We used two datasets, a cars dataset for the Boolean data experiments (Section 6.1), and a publications titles dataset for the text data experiments (Section 6.2). In particular, we use an online used-cars dataset consisting of 15,191 cars for sale in the Dallas area extracted from autos.yahoo.com. In the synthetic workload, each query specifies 1 to 5 peculiarities chosen randomly distributed as follows: 1 attribute – 20%, 2 peculiarities – 30%, 3 peculiarities – 30%, 4 peculiarities – 10%, 5 peculiarities – 10%. We assume that most of the users specify two or three peculiarities.

6.1 Boolean Data

We focus on CB-QR, which can be solved by a superset of the algorithms used in the other variants.

The top-m peculiarities selected by our algorithms seem promising. For example, even with a small real query register of 185 queries, our optimal algorithms could select top features specific to the car, e.g., sporty features are selected for sports cars, safety features are selected for passenger sedans, and so on.

We first compare the execution times of the optimal and greedy algorithms that solve CB-QR. These are (Section 4): ILP-CB-QR, MaxFreqItemSets-CB-QR, which produce optimal results, and ConsumeAttr-CB-QR, ConsumeAttrCumul- CB-QR, and ConsumeQueries-CB-QR, which are greedy approximations. The CB-QR suffix is skipped in the

graphs for clarity. Fig 6 shows how the execution times vary with m for the real query workload averaged over 100 randomly selected to-be-advertised cars from the dataset. Note that different y-axis scales are used for the two optimal and the three greedy algorithms to better display the differences among the methods.

The times in Fig 6 for MaxFreqItemSets also include the preprocessing stage, which can be performed once in advance regardless of the new tuple (user car), as explained in Section 4.3. Fig 7 shows the quality, that is, the numbers of satisfied queries for the greedy algorithms along with the optimal numbers, for varying m. The numbers of queries are averaged over 100 randomly selected to-be-advertised cars from the dataset. Note that no query is satisfied for m = 3 because all queries specify more than 3 peculiarities. We see that ConsumeAttr and ConsumeAttrCumul produce near optimal results. In contrast, ConsumeQueries has low quality, since it is often the case that the peculiarities of the queries with few peculiarities (which are selected first) are not common in the workload.

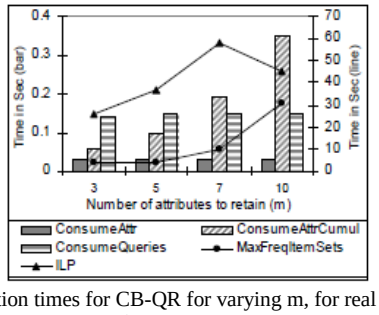

Fig 6: Execution times for CB-QR for varying m, for real workload of 185 queries.

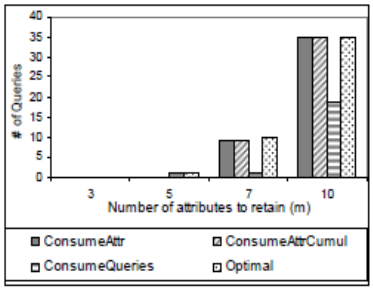

Fig 7: Satisfied queries for greedy and optimal algorithms for CB-QR for varying m, for real workload of 185 queries.

Fig 8 and Fig 9 repeat the same experiments for the synthetic query workload of 2000 queries. In Fig 8, we do not include the ILP algorithm, because it is very slow for more than 1000 queries (as also shown in Fig 10).

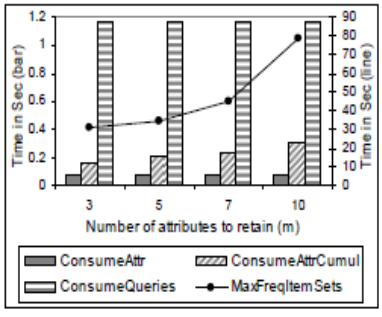

Fig 8: Execution times for CB-QR for varying m, for the synthetic workload of 2000 queries.

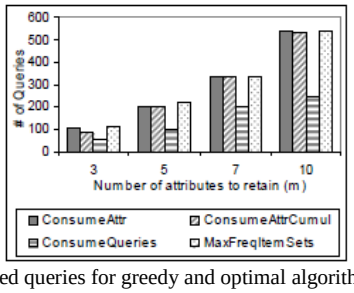

Fig 9: Satisfied queries for greedy and optimal algorithms for CB-QR for varying m, for synthetic workload of 2000 queries.

Next, we measure the execution times of the algorithms for varying query register size and number of peculiarities. Fig 10 shows how the average execution time varies with the query register size, where the synthetic workloads were created as described earlier in this section. We observe that ILP does not scale for large query registers; this is why there are no measurements for ILP for more than 1000 queries. ConsumeQueries performs consistently worse than other greedy algorithms since we make a pass on the whole workload at each iteration to find the next query to add. We conclude ConsumeQueries is generally a bad choice.

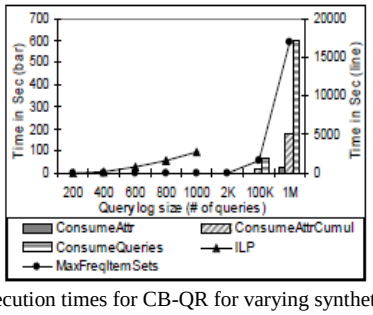

Fig 10: Execution times for CB-QR for varying synthetic workload size for m = 5.

Fig 11 focuses on the two optimal algorithms, and measures the execution times of the algorithms, averaged over 100 randomly selected to-be-advertised cars from the dataset, for varying number M of total peculiarities of the dataset and queries, for a synthetic query register of 200 queries. We observe that ILP is faster than MaxFreqItemSets for more than 32 total peculiarities. For 32 total peculiarities Max-FreqItemSets is faster as also shown in Fig 6. However, note that ILP is only feasible for very small query

registers. For larger query registers, ILP is very slow or infeasible, as is also shown by the missing values in Fig 10.

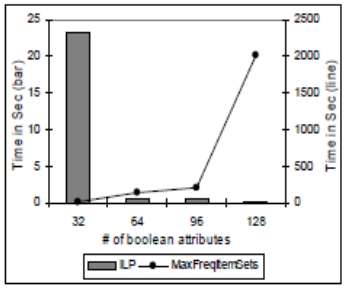

Fig 11: Execution times for CB-QR for varying number of total peculiarities for the synthetic workload of 200 queries for m = 5

To summarize, ILP is better for small query registers and many total peculiarities (i.e. short and wide query register), whereas Max- FreqItemSets is better for larger query registers with fewer total peculiarities (i.e. long and narrow query register). However for query registers those are long and wide, the problem becomes truly intractable, and approximation methods such as our greedy algorithms perhaps the only feasible approaches.

6.2 Screens of Experiment

In this paper, we are describing how a query will be processed. According to that, some of the screens are given here for easy resemblance:

Fig 12: How the buyer is going to observe the peculiarities.

Fig 12 represents when a user logins into the online with the given registered userid and password. Then the user will get the screen with different widgets having the peculiarities.

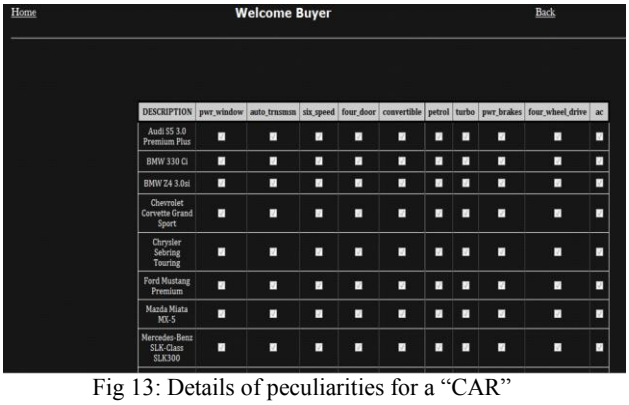

Fig 13: Details of peculiarities for a "CAR"

Fig 13 represents the selection of a widget from the list. In the above list, it represents the peculiarities of the "CAR" widget. So, it gives the specifications of each and every widget.

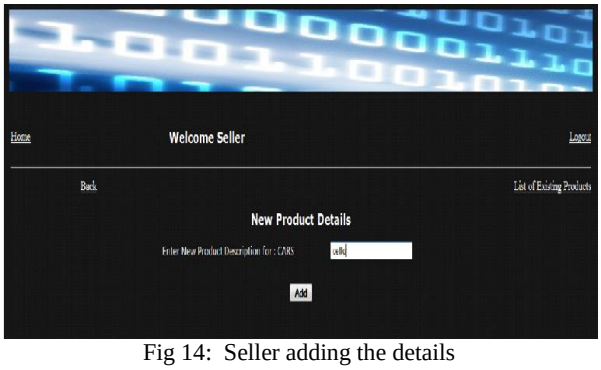

Fig 14: Seller adding the details

Fig 14 represents the screen of the seller for adding the details of a widget. This is very important to seller for updating the peculiarities according to the widgets added.

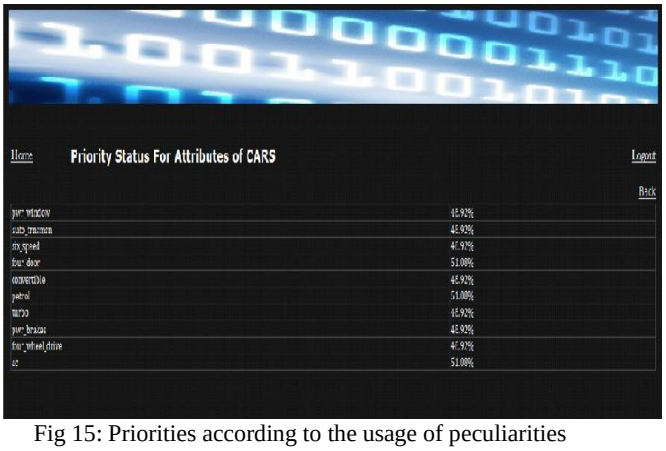

Fig 15: Priorities according to the usage of peculiarities

VII. Associated Works

A large corpus of work has tackled the problem of ranking the results of a query. In the documents world, the most popular procedures are tf-idf based [20] ranking functions, like BM25 [18], as well as link-

structure-based procedures like PageRank [2] if such links are present (e.g., the Web). In the database world, automatic ranking procedures for the results of structured queries have been proposed [1, 7, and 30]. Also there has been recent work [3] on ordering the displayed peculiarities of query results.

Both of these tuple and attribute ranking procedures are inapplicable to our problem. The former inputs a database and a query, and outputs a list of database tuples according to a ranking function, and the latter inputs the list of database results and selects a set of peculiarities that "explain" these results. In contrast, our problem inputs a database, a query register, and a new tuple, and computes a set of peculiarities that will rank the tuple high for as many queries in the query register as possible.

Although the problem of choosing peculiarities is seemingly related to the area of feature selection [5], our work differs from the work on feature selection because our goal is very specific – to enable a tuple to be highly visible to the database users – and not to reduce the cost of building a mining model such as classification or clustering.

Kleinberg et al. [12] present a set of microeconomic problems suitable for data mining procedures; however no specific solutions are presented. Their problem closer to our work is identifying the best parameters for a advertising strategy in order to exploit the attracted customers, given that the competitor independently also prepares a similar strategy. Our problem is different since we know the competition. Another area where boosting an item's rank has received attention is Web search, where the most popular procedures involve manipulating the link structure of the Web to achieve higher perceptibility [8].

Integer and linear programming optimization problems are extremely well studied problems in operations research, management science and many other areas of applicability (see recent book on this subject [19]). Integer programming is well-known to be NP-hard [6]; however carefully designed branch and bound algorithms can capability solve problems of moderate size.

Computing regular itemsets is a popular area of research in data mining and some of the best known algorithms include Apriori [1] and FP-Tree [9]. Several papers have also investigated the problem of computing maximal regular itemsets [3, 5, 12, 14, and 17]. Almost all the popular approaches are designed for sparse datasets and do not work well for our unique problem of dense datasets. Apriori [1] employs a bottom-up, breadth first search that enumerates every single regular itemset. In many applications (especially in dense data) with long regular patterns enumerating all possible subsets of an M length pattern (M can easily be 50 or 60 or longer) is computationally unfeasible. Also, we are not interested in mining all regular itemsets, but only maximal regular itemsets in our algorithm. A known approach for mining maximal regular itemsets is the complete random walk [7], which is a bottom-up approach. To see this, consider a table with 50 peculiarities, and assume we need to determine a compressed tuple t′ with 10 peculiarities. Now, we need to know the itemset of ~Q (complemented query register which is a dense dataset) of size 40 with maximum frequency. Due to the dense nature of ~Q, the bottom-up approach will not be able to compute regular itemsets beyond a size of 5-10. Likewise, other approaches for mining maximal regular itemsets such as the Genetic Algorithm (GA) based approach [11] is also mainly intended for sparse dataset and does not work well for dense dataset. In contrast, our proposed method works well for dense dataset. The recent works [14] and [13] are related to our work. The former tries to find out the dominant relationship between products and potential buyers where by analyzing such relationships, companies can position their products more effectively while remaining profitable. The latter introduces skyline query types taking into account not only min/max peculiarities (e.g., price, weight) but also spatial peculiarities and the relationships between these different attribute types. Their work aims at helping manufacturers choose the right specs for a new product, whereas our work to choose the peculiarities subset of an existing product for advertising purposes.

In previous work [16], we tackled the main variant of the problem with Boolean conjunctive query semantics where a tuple satisfies a query if all the peculiarities present in query are also present in the tuple (Section 3). We extend the idea in the current paper. We consider both the database (existing products) and query register with various query semantics (conjunctive, top-k, skyline, negations, etc.).

Several procedures have been proposed for capable skyline query processing [4, 19, 25, and 31]. There has been recent work on categorical skylines [21] and skyline computation over low cardinality domains [15] that also considers skyline for Boolean data as well. One main difference of our work with the existing works is that our goal is not to propose a method for processing or maintaining the skylines, instead we use skylines as a query semantic where a new tuple can be visible for maximum number of queries. Another related work is mining top-k regular itemsets without minimum support verge [10] which finds top-k closed regular itemsets. Also it is not proven that the top-k approach works well for dense dataset. The top-k approach without minimum support verge [10] finds top-k regular closed patterns of length no less than min_l, where min_l is the minimal length of each pattern.

VIII. CONCLUSIONS

In this work we introduced the problem of picking the best peculiarities of a new tuple, such that this tuple will be ranked highly, given a dataset, a query register,

or both, i.e., the tuple "stands out in the crowd". We presented variants of the problem for Boolean, categorical, text and numeric data, and showed that even though the problem is NP-complete in most cases; optimal algorithms are feasible for small inputs. Furthermore, we present greedy algorithms, which are experimentally shown to produce good approximation ratios. After all, a query register is only an approximate surrogate of real user preferences, and moreover in some applications neither the database, nor the query register may be available for analysis; thus we have to make assumptions about the nature of the competition as well as about the user preferences. Finally, in all these problems our focus is on deciding what subset of peculiarities to retain of a product. We do not attempt to suggest what values to set for specific peculiarities, which is a problem tackled in advertising research, e.g., [17]. However, while we acknowledge that the scope of our problem definition is indeed limited in several ways, we do feel that our work takes an important first step towards developing principled approaches for attribute selection in a data exploration environment.

ACKNOWLEDGMENT

The authors gratefully acknowledge the contributions of the reviewers. Their comments have considerably improved the quality of this article. We, authors express gratitude to all the anonymous reviewers for their affirmative annotations among our paper.

REFERENCES

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky- Shapiro, P. Smyth, and R. Uthurusamy, (eds.), Advances in Information Discovery and Data Mining, pp. 307-328. AAAI/MIT Press, 1996.

[2] S. Brin and L. Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine. WWW Conference, 1998.

[3] Gautam Das, Vagelis Hristidis, Nishant Kapoor, S. Sudarshan. Ordering the Peculiarities of Query Results. SIGMOD, 2006.

[4] Good, I., The population frequencies of species and the estimation of population parameters, Biometrika, v. 40, 1953, pp. 237-264.

[5]Isabelle Guyon and Andre Elisseeff. An introduction to variable and feature selection. Journal of Machine Learning Research, 3(mar): 2003.

[6]Michael R. Garey and David S. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman. ISBN 0-7167-1045-5.

[7] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, R. S. Sharm: Discovering all most specific sentences. ACM TODS. 28(2): 2003

[8] M. Gori and I. Witten. The bubble of web perceptibility. Commun. ACM 48, 3 (Mar. 2005), 115-117.

[9] Jiawei Han, Jian Pei, Yiwen Yin: Mining Regular Patterns without Candidate Generation. SIGMOD 2000: 1-12.

[10]Jiawei Han, Jianyong Wang, Ying Lu, and Petre Tzvetkov: Mining top-k regular closed patterns without minimum support, ICDM 2002.

[11]Jen-peng Huang, Che-Tsung Yang, Chih-Hsiung Fu: A Genetic Algorithm Based Searching of Maximal Regular Itemsets. ICAI 2004.

[12] J. Kleinberg, C. Papadimitriou and P. Raghavan.A Microeconomic View of Data Mining. Data Min. Knowl. Discov. 2, 4 (Dec. 1998).

[13]Cuiping Li, Beng Chin Ooi, Anthony K. H. Tung, Shan Wang: DADA: a Data Cube for Dominant Relationship Analysis. SIGMOD 2006.

[14]Cuiping Li, Anthony K. H. Tung, Wen Jin, Martin Ester: On Dominating Your Neighborhood Profitably. VLDB 2007: 818-829

[15]Michael D. Morse, Jignesh M. Patel, H. V. Jagadish: Capable Skyline Computation over Low-Cardinality Domains. VLDB 2007.

[16] Muhammed Miah, Gautam Das, Vagelis Hristidis, Heikki Mannila: Standing Out in a Crowd: Selecting Attributes for Maximum Visibility. ICDE 2008: 356-365

[17]Thomas T. Nagle, John Hogan. The Strategy and Tactics of Pricing: A Guide to Growing More Profitably (4th Edition), Prentice Hall, 2005.

[18]S E Robertson and S Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted rescue. SIGIR 1994.

[19] Alexander Schrijver: Theory of Linear and Integer Programming. John Wiley and Sons. 1998.

[20]G. Salton. Automatic Text Processing: The Transformation, Analysis, and Rescue of Information by Computer. Addison Wesley, 1989.

[21]Nikos Sarkas, Gautam Das, Nick Koudas, Anthony K. H. Tung: Categorical skylines for streaming data. SIGMOD Conference 2008: 239-250