The VLDB Journal (2013) 22:849–874
DOI 10.1007/s00778-013-0311-4

REGULAR PAPER

YMALDB: exploring relational databases via result-driven recommendations

Marina Drosou · Evaggelia Pitoura

Received: 22 February 2012 / Revised: 8 February 2013 / Accepted: 13 March 2013 / Published online: 17 May 2013

Abstract The typical user interaction with a database system is through queries. However, many times users do not have a clear understanding of their information needs or the exact content of the database. In this paper, we propose assisting users in database exploration by recommending to them additional items, called YMAL ("You May Also Like") results, that, although not part of the result of their original query, appear to be highly related to it. Such items are computed based on the most interesting sets of attribute values, called faSets, that appear in the result of the original query. The interestingness of a faSet is defined based on its frequency in the query result and in the database. Database frequency estimations rely on a novel approach of maintaining a set of representative rare faSets. We have implemented our approach and report results regarding both its performance and its usefulness.

Keywords Recommendations · Faceted search · Data exploration

1 Introduction

Typically, users interact with a database system by formulating queries. This query-response mode of interaction assumes that users are to some extent familiar with the content of the database and that they have a clear understanding of their information needs. However, as databases become larger and accessible to a more diverse and less technically oriented audience, a more exploratory mode of information seeking seems relevant and useful [15].

Previous research has mainly focused on assisting users in refining or generalizing their queries. Approaches to the *many-answers* problem range from reformulating the original query so as to restrict the size of the result, for example, by adding constraints to the query (e.g., [32]), to automatically ranking query results and presenting to users only the top-k most highly ranked among them (e.g., [12]). With *facet search* (e.g., [20]), users start with a general query and progressively narrow its results down to a specific item by specifying at each step facet conditions, i.e., restrictions on attribute values. The *empty-answers* problem is commonly handled by relaxing the original query (e.g., [23]).

In this paper, we propose a novel exploratory mode of database interaction that allows users to discover items that although not part of the result of their original query are highly correlated to this result.

In particular, at first, the interesting parts of the result of the initial user query are identified. These are sets of (attribute, value) pairs, called *faSets*, that are highly relevant to the query. For example, assume a user who asks about the characteristics (such as genre, production year or country) of movies by a specific director, e.g., M. Scorsese. Our system will highlight the interesting aspects of these results, e.g., interesting years, pairs of genre and years, and so on (Fig. 1).

The *interestingness* of each faSet is based on its frequency. Intuitively, the more frequent a faSet in the result, the more relevant to the query. To account for popular faSets, we also consider their frequency in the database. For example, the reason that a movie genre appears more frequently than another may not be attributed to the specific director but to the fact that this is a very common genre. To address the fundamental problem of locating interesting faSets efficiently, we introduce appropriate data structures and algorithms.

M. Drosou (✉) · E. Pitoura
Computer Science Department, University of Ioannina, Ioannina, Greece
e-mail: mdrosou@cs.uoi.gr

E. Pitoura
e-mail: pitoura@cs.uoi.gr

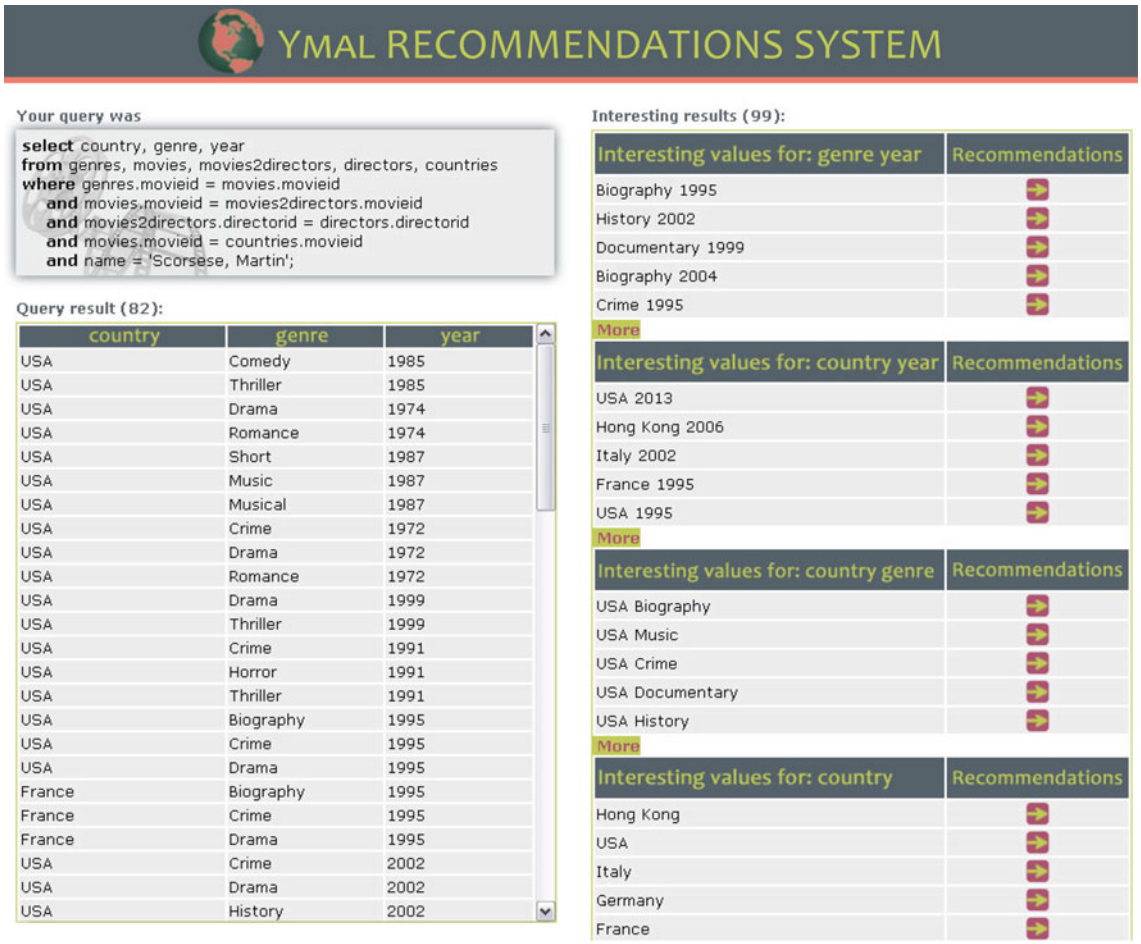

Fig. 1 YMALDB: On the *left side*, the original user query Q is shown at the *top* and its result at the *bottom*. Q asks for the countries, genres and years of movies directed by M. Scorsese. On the *right side*, interesting parts of the result are presented grouped based on their attributes and ranked in order of interestingness

Specifically, since the online computation of the frequency of each faSet in the database imposes large overheads, we maintain an appropriate summary that allows us to *estimate* such frequencies when needed. To this end, we propose a novel approach based on storing the frequencies of a set of representative closed rare faSets. The size of the maintained set is tunable by an ϵ-parameter so as to achieve a desired estimation accuracy. The stored frequencies are then used to estimate the interestingness of the faSets that appear in the result of any given user query. We present a two-phase algorithm for computing the k faSets with the highest interestingness. In the first phase, the algorithm uses the pre-computed summary to set a frequency threshold that is used in the second phase to run a frequent itemset-based algorithm on the result of the query.

After the k most interesting faSets have been located, *exploratory queries* are constructed whose results possess these interesting faSets. The results of the exploratory queries, called YMAL ("You May Also Like") results, are also presented to the user. For example, by clicking on each

Fig. 2 YMALDB: Recommendations for a specific interesting piece of information (Biography films of 1995)

important aspect of the query about movies by M. Scorsese, the user gets additional recommended YMAL results, i.e., other directors who have directed movies with characteristics similar to the selected ones (Fig. 2). This way, users

get to know other, possibly unknown to the them, directors who have directed movies similar to those of M. Scorsese in our example.

Our system, YMALDB, provides users with exploratory directions toward parts of the database that they have not included in their original query. Our approach may also assist users who do not have a clear understanding of the database, e.g., in the case of large databases with complex schemas, where users may not be aware of the exact information that is available.

The offered functionality is complementary to query-response and recommendation systems. Contrary to facet search and related approaches, our goal is not to refine the original query so as to narrow its results. Instead, we provide users with items that do not belong to the results of their original query but are highly related to them. Traditional recommenders [6] and OLAP navigation systems [17] assume the existence of a log of previous user queries or results and recommend items based on the past behavior of this particular user or other similar users. YMAL results are based solely on the database content and the initial query.

We have implemented our approach on top of a relational database system. We present experimental results regarding the performance of our summaries and algorithms using both synthetic and real datasets, namely one dataset containing information about movies [1] and one dataset containing information about automobiles [3]. We have also conducted a user study using the movie dataset and report input from the users.

Paper outline. In Sect. 2, we present our result-driven framework (called REDRIVE) for defining interesting faSets, while in Sect. 3, we use interesting faSets to construct exploratory queries and produce YMAL results. Sections 4 and 5 introduce the summary structures and algorithms used to implement our framework. Section 6 presents our prototype implementation along with an experimental evaluation of the performance and usefulness of our approach. Finally, related work is presented in Sect. 7, and conclusions are offered in Sect. 8.

2 The REDRIVE framework

Our database exploration approach is based on exploiting the result of each user query to identify interesting pieces of information. In this section, we formally define this framework, which we call the REDRIVE framework.

Let $\mathcal{D}$ be a relational database with n relations $\mathcal{R} = \{R_1, \ldots, R_n\}$ and let $\mathcal{A}$ be the set of all attributes in $\mathcal{R}$. We use $\mathcal{A}_C$ to denote the set of categorical attributes and $\mathcal{A}_N$ to denote the set of numeric attributes, where $\mathcal{A}_C \cap \mathcal{A}_N = \emptyset$ and $\mathcal{A}_C \cup \mathcal{A}_N = \mathcal{A}$. Without loss of generality, we assume that relation and attribute names are distinct.

We also define a *selection predicate* c_i to be a predicate of the form $(A_i = a_i)$, where $A_i \in \mathcal{A}_C$ and $a_i \in domain(A_i)$, or of the form $(l_i \leq A_i \leq u_i)$, where $A_i \in \mathcal{A}_N$, $l_i, u_i \in domain(A_i)$ and $l_i \leq u_i$. If $l_i = u_i$, we simplify the notation by writing $(A_i = l_i)$.

To locate items of interest in the database, users pose queries. In particular, we consider select-project-join (SPJ) queries Q of the following form:

SELECT $proj(Q)$
FROM $rel(Q)$
WHERE $scond(Q)$ **AND** $jcond(Q)$

where $rel(Q)$ is a set of relations, $scond(Q)$ is a disjunction of conjunctions of selection predicates, $jcond(Q)$ is a conjunction of join conditions among the relations in $rel(Q)$, and $proj(Q)$ is the set of projected attributes. The *result set*, $Res(Q)$, of a query Q is a relation with schema $proj(Q)$.

2.1 Interesting faSets

Let us first define pieces of information in the result set. We define such pieces, or facets, of the result, as parts of the result that satisfy specific selection predicates.

Definition 1 (m-$FaSet$) An m-faSet, $m \geq 1$, is a set of m selection predicates involving m different attributes.

We shall also use the term faSet when the size of the m-faSet is not of interest.

For a faSet f, we use $Att(f)$ to denote the set of attributes that appear in f. Let t be a tuple from a set of tuples S with schema R; we say that t satisfies a faSet f, where $Att(f) \subseteq R$, if $t[A_i] = a_i$, for all predicates $(A_i = a_i) \in f$ and $l_i \leq t[A_i] \leq u_i$, for all predicates $(l_i \leq A_i \leq u_i) \in f$. We call the percentage of tuples in S that satisfy f, *support* of f in S.

Example. Consider the movies database of Fig. 3 and the query and its corresponding result set depicted in Fig. 4. $\{G.genre = \text{“Biography”}\}$ is a 1-faSet with support 0.375 and $\{1990 \leq M.year \leq 2009, G.genre = \text{“Biography”}\}$ is a 2-faSet with support 0.25.

We are looking for interesting pieces of information at the granularity of a faSet: this may be the value of a single attribute (i.e., a 1-faSet) or the values of m attributes (i.e., an m-faSet).

Example. Consider the example in Fig. 4, where a user poses a query to retrieve movies directed by M. Scorsese. $\{G.genre = \text{“Biography”}\}$ is a 1-faSet in the result that is likely to interest the user, since it is associated with many of the movies directed by M. Scorsese. The same holds for the 2-faSet $\{1990 \leq M.year \leq 2009, G.genre = \text{“Biography”}\}$.

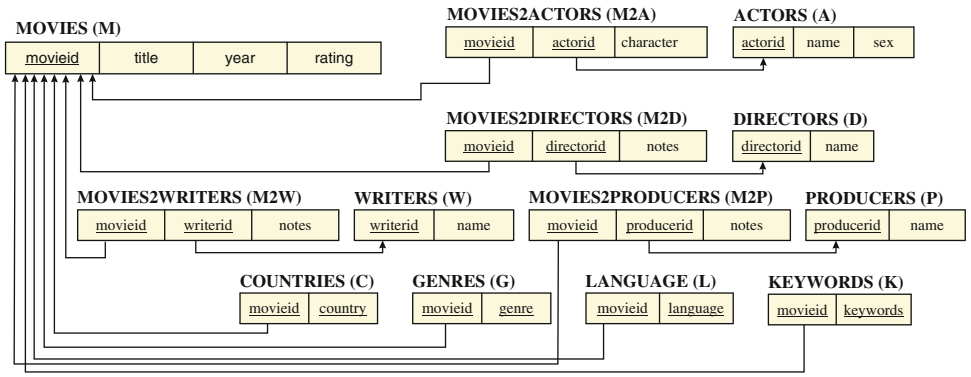

Fig. 3 Movies database schema

```
SELECT D.name, M.title, M.year, G.genre
FROM D, M2D, M, G
WHERE D.name = 'M. Scorsese'
      AND D.directorid = M2D.directorid
      AND M2D.movieid = M.movieid
      AND M.movieid = G.movieid;
```

(a)

D.name	M.title	M.year	G.genre
M. Scorsese	The Aviator	2004	Biography
M. Scorsese	Gangs of New York	2002	Drama
M. Scorsese	Goodfellas	1990	Biography
M. Scorsese	Casino	1995	Drama
M. Scorsese	Shutter Island	2004	Thriller
M. Scorsese	M. Jackson: Video Greatest Hits	1995	Drama
M. Scorsese	The Last Waltz	1978	Biography
M. Scorsese	Raging Bull	1980	Documentary

(b)

Fig. 4 **a** Example query and **b** result set

To define faSet relevance formally, we take an IR-based approach and rank faSets in decreasing order of their odds of being relevant to a user information need. Let u_Q be a user information need expressed through a query Q, and let R_{u_Q} for a tuple t be a binary random variable that is equal to 1 if t satisfies u_Q and 0 otherwise. Then, the relevance of a faSet f for u_Q can be expressed as:

$$\frac{p(R_{u_Q}=1|f)}{p(R_{u_Q}=0|f)}$$

where $p(R_{u_Q}=1|f)$ is the probability that a tuple that satisfies f also satisfies u_Q, and $p(R_{u_Q}=0|f)$ is the probability that a tuple that satisfies f does not satisfy u_Q. Using the Bayes rule we get:

$$\frac{p(R_{u_Q}=1|f)}{p(R_{u_Q}=0|f)} = \frac{p(f|R_{u_Q}=1)p(R_{u_Q}=1)}{p(f|R_{u_Q}=0)p(R_{u_Q}=0)}$$

Since the terms $p(R_{u_Q}=1)$ and $p(R_{u_Q}=0)$ are independent of the faSet f and thus do not affect their relative ranking, they can be ignored.

We make the assumption that all relevant to u_Q tuples are those that appear in $Res(Q)$, thus $p(f|R_{u_Q}=1)$ is equal with the probability that a tuple in the result satisfies f, written $p(f|Res(Q))$. Similarly, $p(f|R_{u_Q}=0)$ is the probability that a tuple that is not relevant, i.e., a tuple that does not belong to the result set, satisfies f. We make the logical assumption that the result set is small in comparison with the size of the database and approximate the non-relevant tuples with all tuples in the database, that is, all tuples in the global relation denoted by $\mathcal{D}$, with schema $\mathcal{A}$. Based on the above motivation, we arrive at the following definition for the relevance of a faSet.

Definition 2 (*interestingness score*) Let Q be a query and f be a faSet with $Att(f) \subseteq proj(Q)$. The interestingness score, $score(f, Q)$, of f for Q is defined as:

$$score(f, Q) = \frac{p(f|Res(Q))}{p(f|\mathcal{D})}$$

The term $p(f|Res(Q))$ is estimated by the support of f in $Res(Q)$, that is, the percentage of tuples in the result set that satisfy f. The term $p(f|\mathcal{D})$ is a global measure that does not depend on the query. It serves as an indication of the frequency of the faSet in the whole dataset, i.e., it measures the discriminative power of f. Note that when the attributes in $Att(f)$ do not belong to the same relation, to estimate this value, we may need to join the respective relations first.

Intuitively, a faSet stands out when it appears more frequently in $Res(Q)$ than anticipated. For a faSet f, $score(f, Q) > 1$, if and only if, its support in the result set is larger than its support in the database, while $score(f, Q) = 1$ means that f appears as frequently as expected, i.e., its support in $Res(Q)$ is the same as its support in the database.

Yet, another way to interpret the interestingness score of a faSet is with relation to the tf-idf (term frequency-inverse

document frequency) measure in IR, which aims to promote terms that appear often in the searched documents but are not very often encountered in the entire corpus. Here, the document roughly corresponds to the result set, the term to a faSet and the corpus to the database.

An association rule interpretation of interestingness. Let r be the current instance of the global database $\mathcal{D}$. r can be interpreted as a transaction database where each tuple constitutes a transaction whose items are the specific (attribute, value) pairs of the tuple. For each query Q, we aim at identifying interesting rules of the form R_f: $scond(Q) \rightarrow f$. In other words, we search for faSets that are highly correlated with the conditions expressed in the user query. Each faSet f is then ranked based on the interestingness or importance of the associated rule R_f. But what makes a rule interesting?

There is large body of research on the topic (see, for example, [25,27,38,39]). For simplicity, let us assume that $Att(proj(Q)) \supset Att(scond(Q))$. Let $count(scond(Q))$ be the number of tuples in r that satisfy $scond(Q)$. Clearly, $count(scond(Q)) = |Res(Q)|$. Common measures of the importance of an association rule are support and confidence, where the *support* of a rule is defined as the percentage of tuples that satisfy both parts of the rule, whereas *confidence* corresponds to the probability that a tuple that satisfies the LHS of the rule also satisfies the RHS. In our case, $support(R_f)$ = (number of tuples in the $Res(Q)$ that satisfy $f)/|\mathcal{D}|$ and $confidence(R_f)$ = (number of the tuples in the result of Q that satisfy $f)/|Res(Q)|$. Using either the support or the confidence of R_f to define the interestingness of faSet f would result in ranking faSets based solely on their frequency in the result set. Note also that for the same number of appearances in the result set, it holds that the larger the result, the smallest the confidence of the rule. This means that more selective queries provide us with rules with higher confidence. However, both measures favor faSets with popular attribute values.

This bias is a known problem of such measures, caused by the fact that the frequency of the RHS of the rule is ignored. This is often demonstrated with the following simple example. Assume that we are looking into the relationship between people who drink tea and coffee, e.g., of a rule of the form *tea* $\rightarrow$ *coffee*. The confidence of such a rule may be high, even when the percentage of people that drink both tea and coffee is smaller than the percentage of the general population of coffee drinkers, as long as this population is large enough.

To handle this problem, another measure of importance for association rules has been introduced, called *lift*, that also accounts for the RHS of the rule so that popular values, or faSets in our case, have to appear more often than less popular ones in the result set to be considered equally important. Lift expresses the probability that a tuple that satisfies the RHS of the rule also satisfies the LHS. We show next that our definition of interestingness for a faSet f corresponds to the lift of rule R_f. Let $p(A)$ be the probability of A appearing in the database. It holds that:

$$\begin{aligned} lift(R_f) &= \frac{p(scond(Q) \wedge f)}{p(scond(Q))P(f)} \\ &= \frac{\frac{count(scond(Q) \wedge f)}{|\mathcal{D}|}}{\frac{count(scond(Q))count(f)}{|\mathcal{D}||\mathcal{D}|}} \\ &= \frac{|\mathcal{D}|}{|Res\,Q|} \frac{count(scond(Q) \wedge f)}{count(f)} \end{aligned}$$

since $|Res(Q)|$ and $|\mathcal{D}|$ are the same for all faSets in the result, lift corresponds to the interestingness measure we use in this paper.

Empty-/Many-answers problem. The goal of our approach is to assist users in exploring a portion of the database that is interesting according to their initial query. This goal is meaningful, when the initial query retrieves a non-empty result set. When the user query retrieves an empty result set, there is no "lead" to point us to possible exploratory directions and the interestingness score of all faSets is zero. In such cases, it is possible to fall back to some default recommendation mechanism or to resort to query relaxation techniques. When $Res(Q)$ contains many answers, the interestingness score still provides us with a means of ranking faSets extracted from these answers. Recall that, we do not aim at narrowing down the initial result of the user query, but rather at locating interesting data related to this result. In this case, the presented faSets can help in highlighting some interesting aspects of this large result set. Note that when the result set has a size comparable to that of the database, one of the assumptions made to motivate the definition of interestingness, namely that the result is small in comparison with the database, may not be valid. However, our definition of interestingness is still valid and provides us with a score based on the relative frequency of each faSet in the result and in the database.

2.2 Attribute expansion

Definition 2 provides a means of ranking the various faSets that appear in the result set, $Res(Q)$, of a query Q and discovering the most interesting ones among them. However, there may be interesting faSets that include attributes that do not belong to $proj(Q)$ and, thus, do not appear in $Res(Q)$. We would like to extend Definition 2 toward discovering such potentially interesting faSets. This can be achieved by *expanding* $Res(Q)$ toward other attributes and relations in $\mathcal{D}$.

Consider, for example, the following query that returns just the titles of movies directed by M. Scorsese in the database of Fig. 3:

SELECT M.title
FROM D, M2D, M
WHERE D.name = 'M. Scorsese'

```
SELECT G.genre, C.country
FROM D, M2D, M, G, C
WHERE D.name = 'M. Scorsese'
  AND M.year > 1963
  AND D.directorid = M2D.directorid
  AND M2D.movieid = M.movieid
  AND M.movieid = G.movieid
  AND M.movieid = C.movieid
```

(a)

```
SELECT D.name, M.year
FROM D, M2D, M, G, C
WHERE G.genre = 'Drama'
  AND C.country = Italy
  AND (D.name <> 'M. Scorsese'
       OR M.year <= 1963)
  AND D.directorid = M2D.directorid
  AND M2D.movieid = M.movieid
  AND M.movieid = G.movieid
  AND M.movieid = C.movieid
```

(b)

```
SELECT D.name, M.year
FROM D, M2D, M, G, C
WHERE G.genre = 'Drama'
  AND C.country = 'Italy'
  AND D.name = 'M. Scorsese'
  AND M.year <= 1963
  AND D.directorid = M2D.directorid
  AND M2D.movieid = M.movieid
  AND M.movieid = G.movieid
  AND M.movieid = C.movieid
```

(c)

```
SELECT D.name, M.year
FROM D, M2D, M, G, C
WHERE G.genre = 'Drama'
  AND C.country = Italy
  AND D.name <> 'M. Scorsese'
  AND M.year > 1963
  AND D.directorid = M2D.directorid
  AND M2D.movieid = M.movieid
  AND M.movieid = G.movieid
  AND M.movieid = C.movieid
```

(d)

Fig. 5 **a** Original user query and **b** the default exploratory query for the interesting faSet {*G.genre = "Drama"*, *C.country = "Italy"*}. **c**, **d** are variations of the default exploratory query; in the former, we recommend M. Scorsese drama movies produced in Italy in different years than those specified in the original user query, and in the latter, we recommend non–M. Scorsese drama movies produced in Italy in the same years as those specified in the original user query

AND D.directorid = M2D.directorid
AND M2D.movieid = M.movieid

All faSets in the result set of Q will appear once (unless M. Scorsese has directed more than one movie with the same title). However, including, for instance, the relation that contains the attribute "Country" in $rel(Q)$ and modifying $jcond(Q)$ accordingly may disclose interesting information, e.g., that many of the movies directed by M. Scorsese are related to Italy.

The definition of interestingness is extended to include faSets with attributes not in $proj(Q)$, by introducing an expanded query Q' with the same selection condition as the original query Q but with additional attributes in $proj(Q')$ and additional relations in $rel(Q')$.

Definition 3 (*expanded interestingness score*) Let Q be a query and f be a faSet with $Att(f) \subseteq \mathcal{A}$. The interestingness score of f for Q is defined as:

$$score(f, Q) = \frac{p(f|Res(Q'))}{p(f|\mathcal{D})}$$

where Q' is an SPJ query with $proj(Q') = proj(Q) \cup Att(f)$, $rel(Q') = rel(Q) \cup \{R' | A_i \in R',$ for $A_i \in Att(f)\}$, $scond(Q') = scond(Q)$ and $jcond(Q') = jcond(Q) \wedge$ (joins with $\{R' \mid A_i \in R',$ for $A_i \in Att(f)\}$).

For instance, expanding our example query toward the "Country" attribute is achieved by the following Q':

```
SELECT M.title, C.country
 FROM D, M2D, M, C
 WHERE D.name = 'M. Scorsese'
  AND D.directorid = M2D.directorid
  AND M2D.movieid = M.movieid
  AND M.movieid = C.movieid
```

We defer the discussion on how we select relations toward which to expand user queries until Sect. 5.3.

3 Exploratory queries

Besides presenting interesting faSets to the users, we use faSets to discover interesting pieces of data that are potentially related to the user needs but do not belong to the results of the original user query. In particular, we construct *exploratory queries* that retrieve results strongly correlated with those of the original user query Q by replacing the selection condition, $scond(Q)$, of Q with equivalent ones, thus allowing new interesting results to emerge. Recall that a high interestingness score for f means that the lift of $scond(Q) \rightarrow f$ is high, indicating replacing $scond(Q)$ with f, since $scond(Q)$ seems to suggest f.

For example, for the interesting faSet {*G.genre = "Drama"*} in Fig. 4, the following exploratory query:

```
SELECT D.name
FROM D, M2D, M, G
WHERE G.genre = 'Drama'
 AND D.name <> 'M. Scorsese'
 AND D.directorid = M2D.directorid
 AND M2D.movieid = M.movieid
 AND M.movieid = G.movieid
```

will retrieve other directors that have also directed drama movies, which is an interesting value appearing in the original query result set. The negation term "D.name <> M. Scorsese" is added to prevent values appearing in the selection conditions of the original user query from being recommended to the users.

Next, we formally define exploratory queries.

Definition 4 (*exploratory query*) Let Q be a user query and f be an interesting faSet for Q. The exploratory query $\hat{Q}$ that uses f is an SPJ query with $proj(\hat{Q}) = Att(scond(Q))$, $rel(\hat{Q}) = rel(Q) \cup \{R' \mid A_i \in R',$ for $A_i \in Att(f)\}$, $scond(\hat{Q}) = f \wedge \neg scond(Q)$ and $jcond(\hat{Q}) = jcond(Q) \wedge$ (joins with $\{R' \mid A_i \in R',$ for $A_i \in Att(f)\}$).

The results of an exploratory query are called YMAL ("You May Also Like") results.

When the selection condition, $scond(Q)$, of the original user query Q contains more than one selection predicate, then instead of just negating $scond(Q)$, we could consider various combinations of these predicates. This means replacing $scond(\hat{Q}) = f \wedge \neg scond(Q)$ in the above definition with

$scond(\hat{Q}) = f \wedge scond(Q) \backslash \{c_i\} \wedge \neg c_i \, c_i \in scond(Q)$. As an example, consider the user query Q of Fig. 5a and assume the interesting faSet {*G.genre* = "*Drama*", *C.country* = "*Italy*"}. Then, the exploratory queries of Fig. 5b–d can be constructed. In general, it is possible to construct up to $2^{|scond(Q)|} - 1$ exploratory queries for each interesting faSet f, each one of them focusing on different aspects of the interesting faSets. In our approach, as a default, we use the exploratory query $\hat{Q}$ where $scond(\hat{Q}) = f \wedge \neg scond(Q)$ for each interesting faSet f. If the users wish to, they can request the execution of other exploratory queries for f as well by specifying combinations of conditions in $scond(Q)$.

The results of an exploratory $\hat{Q}$ are *recommended* to the user. Since in general, the success of recommendations is found to depend heavily on explaining the reasons behind them [40], we include an *explanation* for why each result of $\hat{Q}$ is suggested. The explanation specifies that the presented result appears often with a value that is very common in the result of the original query Q. For example, assuming that F.F. Coppola is a director retrieved by our exploratory query, then the corresponding explanation would be "You may also like F.F. Coppola, since F.F. Coppola appears frequently with the interesting genre Drama and country Italy of the original query."

Clearly, one can use the interesting faSets in the results of an exploratory query to construct other exploratory queries. This way, users may start with an initial query Q and follow the various exploratory queries suggested to them to gradually discover other interesting information in the database. Currently, we do not set an upper limit on the number of exploration steps. Instead, we let users explore the database at the extend they wish, similar to the manner users perform web browsing by following interesting links.

Framework overview. In summary, REDRIVE database exploration works as follows. Given a query Q, the most interesting faSets for Q are computed and presented to the users. Such faSets may be either interesting pieces (subtuples) of the tuples in the result set of Q or expanded tuples that include additional attributes not in the original result. Interesting faSets are further used to construct exploratory queries that lead to discovering additional information, i.e., recommendations, related to the initial user query. Users can explore further the database by exploiting such recommendations for different interesting faSets of the original query or by recursively applying the same procedure on the exploratory queries to retrieve additional interesting faSets and, thus, recommendations.

In the next two sections, we focus on algorithms for the efficient computation of interesting faSets. Note that our algorithms are based on maintaining statistics regarding the frequency of faSets in the database and thus are applicable to any interpretation of interestingness that exploits frequencies.

4 Estimation of interestingness

Let Q be a query with schema $proj(Q)$ and f be an m-faSet with m predicates $\{c_1, \ldots, c_m\}$. To compute the interestingness of f, according to Definition 2 (and Definition 3), we have to compute two quantities: $p(f|Res(Q))$ and $p(f|\mathcal{D})$.

$p(f|Res(Q))$ is the support of f in $Res(Q)$. This quantity is different for each user query Q and, thus, has to be computed online. $p(f|\mathcal{D})$, however, is the same for all user queries. Clearly, the value of $p(f|\mathcal{D})$ for a faSet f could also be computed online. For example, this can be achieved by the following simple count query:

SELECT $count(*)$
FROM $rel(Q)$
WHERE f **AND** $jcond(Q)$

that returns as a result the number of database tuples that satisfy the faSet f. However, one such query is needed for each faSet in $Res(Q)$. Since the number of faSets even for a small $Res(Q)$ is large, this online computation makes the location of interesting faSets prohibitively slow. Thus, we opt for computing offline some information about the frequency of selected faSets in the database and use this information to estimate $p(f|\mathcal{D})$ online. Next, we show how we can maintain such information.

4.1 Basic approaches

Let $m_{\max}$ be the maximum number of projected attributes of any user query, i.e., $m_{\max} = |\mathcal{A}|$. A brute force approach would be to generate all possible faSets of size up to $m_{\max}$ and pre-compute their support in $\mathcal{D}$. Such an approach, however, is infeasible even for small databases due to the combinatorial amount of possible faSets. As an example, consider a database with a single relation R containing 10 categorical attributes. If each attribute takes on average 50 distinct values, R may contain up to $\sum_{i=1}^{10} \left[\binom{10}{i} \times 50^i \right] = 1.1904 \times 10^{17}$ faSets.

A feasible and efficient solution must reach a compromise between the online computation of $p(f|\mathcal{D})$ and the maintenance of frequency information for selected faSets. A first such approach would be to pre-compute and store the support for all 1-faSets that appear in the database. Then, assuming that faSet conditions are satisfied independently from each other, the support of a higher-order m-faSet can be estimated by:

$$p(f|\mathcal{D}) = p\left(\{c_1, \ldots, c_m\}|\mathcal{D}\right) = \prod_{i=1}^{m} p(\{c_i\}|\mathcal{D})$$

This approach requires the storage of information for only a relatively small number of faSets. In our previous example, we only have to maintain information about 10×50

1-faSets. However, although commonly used in the literature, the independence assumption rarely holds in practice and may lead to losing interesting information. Consider, for example, that the 1-faSets {*M.year = 1950*} and {*M.year = 2005*} have similar supports, while the supports of {*G.genre* = "*Sci-Fi*", *M.year = 1950*} and {*G.genre = Sci-Fi*, *M.year = 2005*} differ significantly with {*G.genre* = "*Sci-Fi*," *M.year = 1950*} appearing very rarely in the database. Under the independence assumption, similar estimation values will be computed for these two 2-faSets.

4.2 The closed rare faSets approach

We propose a different form of maintaining frequency summaries, aiming at capturing such fluctuations in the support of related faSets. Our approach is based on maintaining a set of faSets, called ϵ-tolerance closed rare faSets (ϵ-CRFs), and using them to estimate the support of other faSets in the database. Next, we define ϵ-CRFs and show that the estimation error of the support of other faSets is bounded by ϵ, where ϵ is a parameter that tunes the size of the maintained summaries.

Background definitions. First, we define *subsumption* among faSets. We say that a faSet f is subsumed by a faSet f', if every possible tuple in the database that satisfies f also satisfies f'. For example, {*G.genre* = "*Sci-Fi*", $2005 \leq$ *M.year* ≤ 2008} is subsumed by {$2000 \leq$ *M.year* ≤ 2010}. Formally:

Definition 5 (*faSet subsumption*) Let $\mathcal{D}$ be any database and f, f' be two faSets. We say that f is subsumed by f', $f \preceq f'$, if and only if, every possible tuple in the database that satisfies f also satisfies f'.

When $f \preceq f'$, we also say that f is more specific than f' and f' is more general than f. If $f \preceq f'$ and $f' \preceq f$, we say that f and f' are equivalent. f is called a proper more specific faSet of f', denoted $f \prec f'$, if f is subsumed by f' but is not equivalent to it. We also say that f' is a proper more general faSet of f.

Note that, for two faSets f, f' with $f \subseteq f'$, it holds that $f' \preceq f$. For example, {*G.genre = "Sci-Fi"*, $2005 \leq$ *M.year* ≤ 2008} is subsumed by {$2005 \leq$ *M.year* ≤ 2008}.

Following the terminology from frequent itemset mining, given a support threshold ξ_r, we say that a faSet f is *frequent*(FF) for a set of tuples S, if its support in S is greater than or equal to ξ_r and *rare* (RF) if its support is in $[1, \xi_r)$.

We also call a faSet f *closed frequent* (CFF) for S if it is frequent and has no proper more specific faSet f', such that, f' has the same support as f in S. Similarly, we define a faSet f to be *closed rare* (CRF) for S if it is rare and has no proper more general faSet f', such that f' has the same support as f in S.

Finally, we say that a faSet f is *maximal frequent* (MFF) for S, if it is frequent for S and has no more specific faSet f' such that f' is frequent for S and a faSet f is *minimal rare* (MRF) for S if it is rare and has no more general faSet f' such that f' is rare for S.

Summaries based on ϵ-tolerance. Maintaining the support of a number of representative faSets can assist us in estimating the support of a given faSet f. In general, it is more useful to maintain information about the frequency of rare faSets in $\mathcal{D}$, since when rare faSets appear in a result set, it is more likely that they are interesting than when frequent ones do.

Since the number of rare faSets (RFs) may be large, maintaining the support of all rare faSets may not be cost-effective. Minimal rare faSets (MRFs) cannot be maintained either, although their number is small and RFs can be retrieved from MRFs, it is not possible to accurately estimate the support of an RF from MRFs. Instead, closed rare faSets (CRFs) can provide us with both all RFs and their support. Since any RF that has a distinct support value is also a CRF, the number of CRFs may be very close to that of RFs. Thus, in our approach, we maintain a tunable number of CRFs. This number is such that we can achieve a bound on the estimation of the support of any RF as a function of a given parameter ϵ.

We use $count(f, S)$ to denote the absolute number of tuples in a set of tuples S that satisfy a faSet f. We first define the (m, ϵ)-*cover set* of a set of rare m-faSets, or $Cov(m, \epsilon)$, as follows:

Definition 6 ($Cov(m, \epsilon)$) A set of m-faSets is called an (m, ϵ)-cover set for a set of tuples S, denoted $Cov(m, \epsilon)$, if (1) all its faSets are satisfied by at least one tuple in S, (2) for every rare m-faSet f in S, there exists a more general rare m-faSet $f' \in Cov(m, \epsilon)$ with $count(f', S) \leq (1 + \epsilon)\ count(f, S)$, where $\epsilon \geq 0$, and (3) it has no proper subset for which the above two properties hold.

In the following, we seek to locate (m, ϵ)-cover sets that are minimum, i.e., there is no other (m, ϵ)-cover set for the same set of faSets that has a smaller size.

We say that a faSet f' ϵ-subsumes a faSet f, if $f \preceq f'$ and $count(f', S) \leq (1 + \epsilon)\ count(f, S)$.

Example. Consider the attribute *M.year* of the database in Fig. 3 and let us focus, for illustration purposes, on a simple example concerning the movies produced from 1960 to 1990. Assume that there are 10 movies produced in the 60s, 10 movies produced in the 70s and 20 movies produced in the 80s. Consider the 1-faSets {$1960 \leq$ *M.year* ≤ 1970}, {$1960 \leq$ *M.year* ≤ 1980}, {$1960 \leq$ *M.year* ≤ 1990}, {$1970 \leq$ *M.year* ≤ 1980}, {$1970 \leq$ *M.year* ≤ 1990} and {$1980 \leq$ *M.year* ≤ 1990} with counts 10, 20, 40, 10, 30 and 20, respectively. Let also $\epsilon = 1.0$. Then, {$1960 \leq$ *M.year* ≤ 1980} ϵ-subsumes {$1960 \leq$ *M.year* ≤ 1970} and {$1970 \leq$ *M.year* ≤ 1980}, {$1970 \leq$ *M.year* ≤ 1990} ϵ-subsumes {$1980 \leq$ *M.year* ≤ 1990} and {$1960 \leq$ *M.year* ≤ 1990} ϵ-subsumes {$1980 \leq$ *M.year* ≤ 1990}, {$1960 \leq$ *M.year* ≤ 1980} and {$1970 \leq$ *M.year* ≤ 1990} (Fig. 6). The sets {{$1960 \leq$ *M.year*

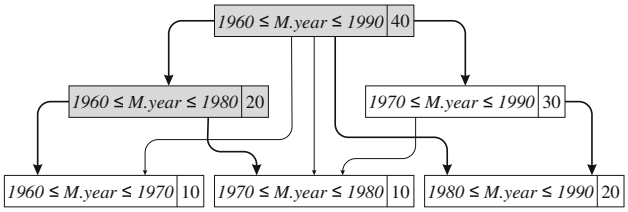

Fig. 6 Example of a minimum (m, ϵ)-cover set (depicted in gray) for the faSets depicted here ($m = 1$) along with their counts for $\epsilon = 1.0$. Arrows represent subsumption relations and bold arrows represent ϵ-subsumption relations

Algorithm 1 Locating an (m, ϵ)-cover set.

Input: A set of m-faSets X, ϵ.
Output: An (m, ϵ)-cover set for X.

```
1: begin
2: Y ← ∅
3: while at least one faSet in X ε-subsumes another do
4:    pick the faSet f' ∈ X that ε-subsumes the largest number of
      faSets in X
5:    for each such faSet f do
6:       merge f with f'
7:       X ← X\{f}
8:    end for
9:    X ← X\{f'}
10:   Y ← Y ∪ {f'}
11: end while
12: return Y
13: end
```

$\leq 1980\}, \{1960 \leq M.year \leq 1990\}\}$ and $\{\{1960 \leq M.year \leq 1970\}, \{1970 \leq M.year \leq 1980\}, \{1960 \leq M.year \leq 1990\}\}$ are both (1, 1.0)-cover sets for this set of faSets, since they both cover all faSets. The former is also a minimum set with this property.

An (m, ϵ)-cover set is, intuitively, the smallest set that can represent all faSets of the same size if we allow the counts of the faSets being represented to differ up to a scale of $(1 + \epsilon)$ from the count of the faSet that represents them. The problem of locating (m, ϵ)-cover sets is an NP-hard problem, similar to the case of the SET COVER problem. We can use a greedy heuristic to locate sub-optimal (m, ϵ)-cover sets. Locating sub-optimal (m, ϵ)-cover sets affects only the size of the summaries we maintain and not the bound of the estimations they provide. In this paper, we use the greedy heuristic shown in Algorithm 1; at each round, we select to add to $Cov(m, \epsilon)$ the faSet f' that ϵ-subsumes the largest number of other faSets and ignore those other faSets from further consideration.

Cover sets allow us to group together faSets of the same size. To group together faSets of different sizes, we build upon the notion of δ-tolerance closed frequent itemsets [13] and define ϵ-CRFs as follows:

Definition 7 (ϵ-*CRF*) An m-faSet f is called an ϵ-CRF for a set of tuples S, if and only if, $f \in Cov(m, \epsilon)$ for S and it has no proper more general rare faSet f' with $|f| - |f'| = 1$ and $f' \in Cov(m - 1, \epsilon)$, such that $count(f', S) \leq (1 + \epsilon)\ count(f, S)$, where $\epsilon \geq 0$.

Intuitively, a rare m-faSet f is an ϵ-CRF if, even if we increase its count by a constant ϵ, all the $(m - 1)$-faSets that subsume it still have a larger count than f. This means that f has a significantly different count from all its more general faSets and cannot be estimated (or represented) by any of them.

Let us assume that a set of ϵ-CRFs is maintained for some value of ϵ. We denote this set C. An RF f either belongs to C or not. If $f \in C$, then the support of f is stored and its count is readily available. If not, then, according to Definitions 6 and 7, there is some faSet that subsumes f that belongs to C whose support is close to that of f. Therefore, given an RF f, we can estimate its count based on its closest more general faSet in C. If there are many such faSets, we use the one with the smallest count, since this can estimate the count of f more accurately. We use $C(f)$ to denote the faSet in C that is the most suitable one to estimate the count of f. The following lemma holds:

Lemma 1 *Let C be a set of ϵ-CRFs for a set of tuples S and f be an RF for S, $f \notin C$. Then, there exists f', $f' \in C$ with $|f| - |f'| = i$, such that, $count(f', S) \leq \phi\ count(f, S)$, where $\phi = (1 + \epsilon)^{2i+1}$.*

Proof Let f be a faSet of size m and C the set of maintained ϵ-CRFs. If $f \notin C$, then, according to Definition 6, there exists an m-faSet f_1, such that, $count(f_1, S) \leq (1 + \epsilon)\ count(f, S)$. If $f_1 \notin C$, then, according to Definition 7, there exists an $(m - 1)$-faSet f_2, such that, $count(f_2, S) \leq (1+\epsilon)\ count(f_1, S)$ and so on. At some point, we will reach a faSet f' that belongs in C. Let $|f| - |f'| = i$. To reach this faSet, we have made at most $i + 1$ steps between faSets of the same size and at most i steps between faSets of different size, and thus, the lemma holds. □

To provide more accurate estimations, each ϵ-CRF f is stored along with its *frequency extension*, i.e., a summary of the actual frequencies of all the faSets that f represents. Recall that, an ϵ-CRF f may represent faSets of different sizes, as indicated by Lemma 1. The frequency extension of an ϵ-CRF is defined as follows.

Definition 8 (*frequency extension*) Let C be a set of ϵ-CRFs for a set of tuples S and f be a faSet in C. Let also $\mathcal{X}(f)$ be the set of all RFs represented in C by f. Then, $X_i(f) = \{x | x \in \mathcal{X}(f) \wedge |x| - |f| = i\}$, $0 \leq i \leq m$, where $m = \max\{i | X_i(f) \neq \emptyset\}$. The frequency extension of f for i, $0 \leq i \leq m$, is defined as:

$$ext(f, i) = \frac{\sum_{x \in X_i(f)} \frac{count(x,S)}{count(f,S)}}{|X_i(f)|}$$

Intuitively, the frequency extension of f for i is the average count difference between f and all the faSets that f represents whose size difference from f is equal to i. Given a faSet f, the estimation of $p(f|\mathcal{D})$, denoted $\tilde{p}(f|\mathcal{D})$, is equal to:

$$\tilde{p}(f|\mathcal{D}) = count(C(f), S) \cdot ext(C(f), |f| - |C(f)|)$$

It holds that

Lemma 2 *Let f be an ϵ-CRF. Then, for each i, it holds that $\frac{1}{\phi} \leq ext(f, i) \leq 1$, where $\phi = (1+\epsilon)^{2i+1}$.*

Proof At one extreme, all faSets in $X_i(f)$ have the same count as f. Then, $\forall x \in X_i(f)$, it holds that $count(x, S) = count(f, S)$ and $ext(f, i) = 1$. At the other extreme, all faSets in $X_i(f)$ differ as much as possible from f. Then, $\forall x \in X_i(f)$, it holds that $count(f, S) = \phi\, count(x, S)$ and $ext(f, i) = 1/\phi$. □

Similar to the proof in [13], it can be shown that the estimation error is bounded by ϕ, i.e., by ϵ.

Theorem 1 *Let f be an RF and $|f| - |C(f)| = i$. The estimation error for $p(f|\mathcal{D})$ is bounded as follows:*

$$\frac{1}{\phi} - 1 \leq \frac{\tilde{p}(f|\mathcal{D}) - p(f|\mathcal{D})}{p(f|\mathcal{D})} \leq \phi - 1$$

Proof From Lemma 2, it holds that $\frac{p(C(f)|\mathcal{D})}{\phi} \leq p(C(f)|\mathcal{D}) \times ext(C(f), i) \leq p(C(f)|\mathcal{D})$. Since $\tilde{p}(f|\mathcal{D}) = count(C(f), S) \times ext(C(f), i)$, it holds that $\frac{p(C(f)|\mathcal{D})}{\phi} \leq \tilde{p}(f|\mathcal{D}) \leq p(C(f)|\mathcal{D})$ (1). Also, it holds that $\frac{p(C(f)|\mathcal{D})}{\phi} \leq p(f|\mathcal{D})$ and, since $f \preceq C(f)$, $p(f|\mathcal{D}) \leq p(C(f)|\mathcal{D})$. Therefore, $\frac{p(C(f)|\mathcal{D})}{\phi} \leq p(f|\mathcal{D}) \leq p(C(f)|\mathcal{D})$ (2). From (1), (2) the theorem holds. □

Tuning ϵ. Parameter ϵ bounds the estimation error for the frequencies of the various faSets. Smaller ϵ values lead to better frequency estimations. However, this comes at the price of increased storage requirements, since in the case of smaller ϵ values, more faSets enter the set of ϵ-CRFs and, therefore, the size of the maintained statistics increases. Next, we provide a method to assist the system administrator in deciding an appropriate ϵ value, given a maximum storage budget b available for maintaining statistics.

Our basic idea is to start with a rough estimation of ϵ and then further refine it to reach the minimum ϵ value that can provide statistics which can fit in the allocated storage space. Our initial estimation is computed as follows. Let $MGF(f)$ be the set of more general proper faSets of a faSet f, i.e., $MGF(f)$ includes all faSets f' that are more general than f with $|f| - |f'| = 1$. We define $g(f)$ to be the average count difference between f and the faSets in $MGF(f)$, i.e, $g(f) = (1/|MFG(f)|) \sum_{f' \in MFG(f)} \frac{count(f', S)}{count(f, S)}$. Then, we define the set of all rare faSets in S as $RF(S)$ and set the initial value of ϵ, denoted ϵ_0 to be equal to $(1/|RF(S)|) \sum_{f \in RF(S)} g(f) - 1$.

We proceed as follows. Let ϵ_0 be that initial value. We use ϵ_0 to locate ϵ_0-CRFs. If the number of located faSets is larger than the maximum allowed threshold, we set $\epsilon_1 = 2\epsilon_0$, otherwise we set $\epsilon_1 = \epsilon_0/2$ and we locate ϵ_1-CRFs. We repeat this process until we reach the first value of ϵ_i that crosses the storage boundary. ϵ_{i-1} and ϵ_i can be used as upper and lower bounds for the final estimation, since it holds either $|\epsilon_i$-CRFs$| > b$ and $|\epsilon_{i-1}$-CRFs$| \leq b$ or vice versa. We set $\epsilon_{i+1} = (\epsilon_{i-1} + \epsilon_i)/2$, update either the upper or lower bound, respectively, and repeat this binary search process until either $|\epsilon_{i+1}$-CRFs$| = b$ or $|\epsilon_{i+1}$-CRFs$| = |\epsilon_i$-CRFs$|$.

In the above process, we generate all rare faSets once and then we proceed with multiple generations of ϵ-CRFs. As shown in our performance evaluation, the cost of generating statistics is dominated by the cost of generating all rare faSets, while the cost of locating ϵ-CRFs is negligible in comparison.

Estimation overview. Given a threshold ξ_r and a value for ϵ, we maintain the set of ϵ-CRFs along with the corresponding frequency extensions. This set, whose size can be tuned by varying ϵ, provides us with bounded estimations of $p(f|\mathcal{D})$ for all rare faSets, that is, for all faSets with support smaller than ξ_r. For frequent faSets, we have only the information that their support is larger than ξ_r, but this in general suffices, since it is not likely that these faSets are interesting.

5 Top-*k* faSets computation

In this section, we present an online two-phase algorithm for computing the top-k most interesting faSets for a user query Q. We consider first faSets f in the result set, i.e., $Att(f) \subseteq proj(Q)$ and discuss attribute expansion later. A straightforward method would be to generate all faSets in $Res(Q)$, compute their interestingness score and then selecting the best among them. This approach, however, is exponential on the number of distinct values that appear in $Res(Q)$. Applying an a priori approach for generating and pruning faSets is not applicable either, since the interestingness score is neither an upwards nor a downwards closed measure, as shown below. A function d is *monotone* or *upwards closed* if for any two faSets f_1 and f_2, $f_2 \preceq f_1 \Rightarrow d(f_1) \leq d(f_2)$ and *anti-monotone* or *downwards closed* if $f_2 \preceq f_1 \Rightarrow d(f_1) \geq d(f_2)$.

Proposition 1 *Let Q be a query and f be a faSet. Then, $score(f, Q)$ is neither an upwards nor a downwards closed measure.*

Proof Let f_1, f_2, f_3 be three faSets with $f_1 \preceq f_2 \preceq f_3$. Consider a database consisting of a single relation R with three attributes A, B and C and three tuples $\{1, 1, 1\}, \{1, 1, 2\}, \{1, 2, 1\}$. Let $Res(Q) = \{\{1, 1, 1\}, \{1, 2, 1\}\}$ and $f_1 = \{A = 1, B = 1, C = 1\}$, $f_2 = \{A = 1, B = 1\}$ and $f_3 = \{A = 1\}$. For f_2, there exists both a more general faSet, i.e., f_3, and a more specific faSet, i.e., f_1, with

Algorithm 2 Two-Phase Algorithm (TPA).

Input: Q, $Res(Q)$, k, C, ξ_r of C.
Output: The top-k interesting faSets for Q.

```
1: begin
2: F ← ∅
3: A ← all 1-faSets of Res(Q)
4: for all faSets f ∈ C do
5:    if all 1-faSets g ⊆ f are contained in A then
6:       f.score = score(f, Q)
7:       F ← F ∪ {f}
8:    end if
9: end for
10: for all tuples t ∈ Res(Q) do
11:    generate all faSets f ⊆ t, s.t. ∃ g ∈ F with g ⊆ f
12:    for all such faSets f do
13:       f.score = score(f, Q)
14:       F ← F ∪ {f}
15:    end for
16: end for
17: ξ_f ← (k^th highest score in F) × ξ_r
18: candidates ← frequentFaSetMiner(Res(Q), ξ_f)
19: for all faSets f in candidates do
20:    f.score = score(f, Q)
21:    F ← F ∪ {f}
22: end for
23: return The k faSets in F with the highest scores
24: end
```

larger interestingness scores than it. The interestingness score is not closed even for the case of faSets of the same size. For example, consider the relation R' with a single attribute A and three tuples $\{1\}, \{3\}, \{4\}$ and $Res(Q) = \{\{1\}, \{4\}\}$ and let $f_1 = \{0 \leq A \leq 10\}$, $f_2 = \{2 \leq A \leq 8\}$, $f_3 = \{4 \leq A \leq 5\}$. Again, for f_2, there exists both a more general and a more specific faSet with larger interestingness score than it.

This implies that we cannot employ any subsumption relations among the faSets of $Res(Q)$ to prune the search space.

5.1 The two-phase algorithm

To avoid generating all faSets in $Res(Q)$, as a baseline approach, we consider only the frequent faSets, since these are the faSets of potential interest. To generate all frequent faSets, i.e., all faSets whose support in $Res(Q)$ is above a given threshold ξ_f, we apply an adaptation of a frequent itemset mining algorithm [19] such as the Apriori or FP-Growth. Then, for each frequent faSet f, we use the maintained summaries to estimate $p(f|\mathcal{D})$ and compute $score(f, Q)$.

The problem with the baseline approach is that it is highly dependent on the support threshold ξ_f. A large value of ξ_f may lead to losing some less frequent in the result but very rarely appearing in the dataset faSets, whereas a small value may result in a very large number of candidate faSets being examined. Therefore, we propose a Two-Phase Algorithm (TPA), described next, that addresses this issue by setting ξ_f to an appropriate value so that all top-k faSets are located without generating redundant candidates. The TPA assumes that the maintained summaries are based on keeping rare faSets of the database $\mathcal{D}$. Let ξ_r be the maximum support of the maintained rare faSets.

In the first phase of the algorithm, all 1-faSets that appear in $Res(Q)$ are located. The TPA checks which rare faSets of $\mathcal{D}$, according to the maintained summaries, contain only conditions that are satisfied by at least one tuple in $Res(Q)$. Let F be this set of faSets. Then, in one pass of $Res(Q)$, all faSets of $Res(Q)$ that are more specific than some faSet in F are generated and their support in $Res(Q)$ is measured. For each of the located faSets, $score(f, Q)$ is computed. Let s be the k^{th} highest score among them. The TPA sets ξ_f equal to $s \times \xi_r$ and proceeds to the second phase where it executes a frequent faSet mining algorithm with threshold equal to ξ_f to retrieve any faSets that are potentially more interesting than the k^{th} most interesting faSet located in the first phase.

Theorem 2 *The Two-Phase Algorithm retrieves the top-k most interesting faSets.*

Proof It suffices to show that any faSet in $Res(Q)$ less frequent than ξ_f clearly has interestingness score smaller than s, i.e., the score of the k^{th} most interesting faSet located in the first phase and, thus, can be safely ignored. To see this, let f be a faSet examined in the second phase of the algorithm. Since the score of f has not been computed in the first phase, then $p(f|\mathcal{D}) > \xi_r$. Therefore, for $score(f, Q) > s$ to hold, it must be that $p(f|Res(Q)) > s \times p(f|\mathcal{D})$, i.e., $p(f|Res(Q)) > s \times \xi_r$.

The TPA is shown in Algorithm 2, where we use C to denote the collection of maintained summaries.

5.2 Improving performance

Next, we discuss a number of improvements concerning the performance of summaries generation and the TPA.

Discretization of numeric values. The cost of generating summaries and executing the TPA mostly depends on the number of distinct attribute values that appear in the database. The higher this number is, the more faSets have to be generated and have their frequencies computed. To reduce the computational cost of our approach, we consider further summarizing numeric attribute values by partitioning the domain space of numeric attributes into non-overlapping intervals and replacing each value in the database by the corresponding interval of values close to it. Similar techniques for domain partitioning have been used in the field of data mining. As in [35], which considers the problem of mining association rules in the presence of both categorical and numeric attributes, we follow the approach of splitting the domain of numeric attributes into intervals and mapping each value to the corresponding interval prior to processing our data. The intervals are chosen in different ways for

each attribute, depending on the semantical meaning of the attribute or the distribution of values. For example, in case of attributes containing information such as years and ages, the intervals correspond to decades. It is possible to follow a similar approach for categorical attributes as well by grouping attribute values based on some hierarchy. However, this requires the knowledge of such hierarchies which are not usually available. In our work, we do not further consider grouping categorical values.

Exploiting Bloom filters for fast frequency estimations. Most real datasets have a large number of rare faSets that appear only once. Consider, for example, the movies database of Fig. 3, where many directors have directed only one movie in their lifetime and, therefore, they appear only once in the database. Although such values may have high interestingness score, since they are extremely rare in the whole dataset, they are not useful for recommending additional results to the users. To see this, let us assume that a user queries the database for Sci-Fi movies and a director who appears only once in the dataset is found in the result. Our framework would attempt to recommend to the user other genres that this specific director has directed. However, since this director appears only once in the database, no such recommendations can emerge.

To avoid generating and maintaining information for all other faSets that these rare faSets subsume, we use the following approach. In a single scan of the data, we identify all faSets that appear only once and insert them in a hash-based data structure. In particular, we use a Bloom filter [8]. A Bloom filter consists of a bit array of size l and a set of h hash functions. Each of the hash functions maps a value to one of the l positions of the bit array. To add a value into the Bloom filter, the value is hashed using each of the hash functions and the h corresponding bits are set to 1. To decide whether a value has been added into the Bloom filter, the value is again hashed using each of the hash functions and the corresponding h bits are checked. If all of them are set to 1, then it can be concluded that the value has been added into the Bloom filter. It is possible that those h bits were set to 1 during the insertion of other values; in that case, we have a false positive. It is known that, when n values have been inserted into a Bloom filter, the probability of a false positive is equal to $(1-e^{-hn/l})^h$ and, thus, can by tuned by choosing an appropriate size l for the Bloom filter.

Any faSet that is subsumed by some faSet in the Bloom filter can appear only once in the database. We exploit this fact in two ways. First, we avoid the generation and maintenance of ϵ-CRFs that are subsumed by faSets in the Bloom filter, maintaining only the Bloom filter instead which is more space efficient and can support faSet lookups faster. More specifically, whenever a candidate rare faSet f is constructed during the generation of the summaries, we query the Bloom filter for any sub-faSet of f. In case such sub-faSets exist, then f cannot appear more than once in the database and, thus, f is also inserted in the Bloom filter and pruned from further consideration. Second, during the candidate generation phase of the TPA, we also prune candidates that are subsumed by some faSet in the Bloom filter, thus reducing the computational cost of the algorithm. The frequency of those faSets can be estimated as being equal to 1.

We have also generalized the use of Bloom filers for pruning more faSets during the generation of ϵ-CRFs. In particular, we also insert into the Bloom filter all faSets with frequency below some small system defined threshold value ξ_0, with $\xi_0 < \xi_r$.

Using random walks for the generation of rare faSets. Our approach is based on the generation of all ϵ-CRFs for a given threshold ξ_r. A number of different algorithms exist in the literature on which this generation can be based (e.g., [37]). Generally, the generation of all CRFs and even RFs is required as an intermediate step in most cases. However, locating all respective RFs for large datasets becomes inefficient, due to the exponential nature of algorithms such as Apriori. To overcome this, we use a *random walks*-based approach [18] to generate RFs. In particular, we do not produce all RFs as an intermediate step for computing ϵ-CRFs but, instead, we produce only a subset of them discovered by random walks initiated at the MRFs. Our experimental results indicate that, even though not all RFs are generated, we still achieve good estimations for the frequencies of the various faSets.

5.3 FaSet expansion

For a query Q, following the discussion of Sect. 2.2, besides considering the faSets whose attributes belong to $proj(Q)$, we would also like to consider potentially interesting faSets that have additional attributes. Clearly, considering all possible faSets for all combinations of (attribute, value) pairs is prohibitive. Instead, we consider adding to $proj(Q)$ a few additional attributes $\mathcal{B}$ that appear relevant to it. Then, we construct and execute Q' as defined in Definition 3 and use the TPA to compute the top-k (expanded) most interesting faSets of Q'.

The selection of these attributes is dictated by *expansion rules*. An expansion rule is a rule of the form $A \rightarrow B$, where A is a set of attributes in the user query, i.e., $A \subseteq proj(Q)$ and B is a set of attributes in the database, i.e., $B \subseteq \mathcal{A} \backslash proj(Q)$. The meaning of an expansion rule is that when a query Q contains all attributes of A in its select clause, then it should be expanded to contain the attributes of B as well. The attributes of B do not necessarily belong to the relations of $rel(Q)$. Let $A^1, \ldots, A^r$ be attributes of $proj(Q)$ and

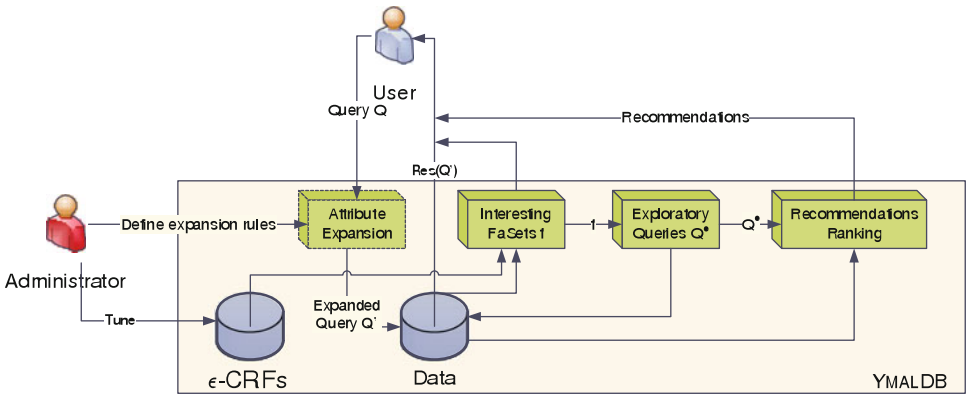

Fig. 7 The system architecture of YMALDB

$A^1 \rightarrow B^1, \ldots, A^r \rightarrow B^r$ be the corresponding applicable expansion rules. Then, $\mathcal{B} = \cup_{i=1}^{r} B^i$.

Our default approach to faSet expansion is to expand each user query Q toward one relation from the database $\mathcal{D}$. We consider only the relations that are adjacent to the query Q, i.e., have a foreign key connection to some relation in $rel(Q)$. From these adjacent relations, we choose the one that is "mostly connected" with $rel(Q)$, i.e., the one for which the size of its join with its adjacent relation in $rel(Q)$ is the largest. The main reason for this is that the relation with the largest size of join will offer more database tuples, and therefore, more interesting faSets may be located. We expand Q toward all the non-id attributes from the relation that was selected as described above.

6 Experimental results

In this section, we first present YMALDB, our prototype recommendation system. Then, we present experimental results regarding the efficiency of our approach. We conclude the section with a user study.

6.1 YMALDB

YMALDB is implemented in Java (JDK 1.6) on top of MySQL 5.0. Our system architecture is shown in Fig. 7. After the user submits a query, an optional query expansion step is performed. Then, the query results along with the maintained ϵ-CRFs are exploited to locate interesting faSets in the result. These faSets are presented to the user who can request the execution of exploratory queries for any of the presented faSets and retrieve the corresponding recommendations.

We next describe the user interface and information flow in YMALDB in more detail. YMALDB can be accessed via a simple web browser using an intuitive GUI. Users can submit their SQL queries and see recommendations, i.e., YMAL results. Along with the results of their queries, users are presented with a list of interesting faSets based on the query result (Fig. 1). Since the number of interesting faSets may be large, interesting faSets are grouped in categories according to the attributes they contain. Larger faSets (i.e., faSets that include more attributes) are presented higher in the list, since larger faSets are in general more informative. The faSets in each category are ranked in decreasing order of their interestingness score and the top-5 faSets of each category are displayed. Additional interesting faSets for each category can be displayed by clicking on a "More" button. We also present the top-5 faSets with the overall best interestingness score independent of the category they belong to.

An arrow button appears next to each interesting faSet. When the user clicks on it, a set of YMAL results, i.e., recommendations, appear (Fig. 2). These recommendations are retrieved by executing an exploratory query for the corresponding faSet. An explanation is also provided explaining how these specific recommendations are related to the original query result. Users are allowed to turnoff the explanation feature.

Since the number of results for each exploratory query may be large, these results are ranked. Many ranking criteria can be used. In our current implementation, we present the results ranked based on a notion of *popularity*. Popularity is application-specific, for example, in our movies dataset, when the YMAL results refer to people, such as directors or actors, we use the average rating of the movies in which they participate and present recommendations in descending order of the associated rank. We present the top-10 recommendations for each faSet. If users wish to do so, they can request to see more recommendations.

Furthermore, users may ask to execute more exploratory queries. This can be achieved by either (a) recursively, i.e., treating the exploratory query as a regular query and finding interesting faSets in its result, or (b) relaxing the negation of the exploratory query, i.e., relaxing some of the selection conditions of the original query.

Finally, users may request the expansion of their original queries with additional attributes. Instead of automatically performing attribute expansion, expansion is done only if requested explicitly, to avoid confusing the users with unrequested attributes. The results of the original user query are expanded toward the set of attributes indicated by the expan-

sion rules. Users receive a list of interesting faSets and recommendations as before.

We have also provided an administrator interface to allow the fine tuning of the various performance-related parameters (i.e., ϵ, ξ_r and ξ_0) and also the specification of additional expansion rules if needed.

In our user study in Sect. 6.4, we evaluate many of the design decisions regarding the presentation of interesting faSets and recommendations as well as regarding explanations and expansions.

6.2 Datasets

We use both real and synthetic datasets. Synthetic datasets consist of single relations, where each attribute takes values from a zipf distribution with parameter θ. We use 10,000 tuples and 7 or 10 attributes for each relation. We also experiment with different values of θ (we report results for $\theta = 1.0$ and $\theta = 2.0$). We use "ZIPF-$|\mathcal{A}|$-θ" to denote a synthetic dataset with $|\mathcal{A}|$ attributes and zipf parameter θ. We also use two real databases. The first one ("AUTOS") is a single-relation database consisting of 12 characteristics for 15,191 used cars from Yahoo!Auto [3]. We also use a subset of this dataset containing 7 of these characteristics. The second one ("MOVIES") is a multi-relation database containing information extracted from the Internet Movie Database [1]. The schema of this database is shown in Fig. 3. The cardinality of the various relations ranges from around 10,000 to almost 1,000,000 tuples. We report results for a subset of relations, namely Movies, Movies2Directors, Directors, Genres and Countries.

6.3 Performance evaluation

We start by presenting performance results. There are two building blocks in our framework. The first one is a precomputation step that involves maintaining information, or summaries, for estimating the frequency of the various faSets in the database. The second one involves the run-time deployment of the maintained information in conjunction with the results of the user query toward discovering the k most interesting faSets for the query. Next, we evaluate the efficiency and the effectiveness of these two blocks.

We executed our experiments on an Intel Pentium Core2 2.4 GHz PC with 2 GB of RAM.

6.3.1 Generation of ϵ-CRFs

We evaluate the various options for maintaining rare faSets in terms of (1) storage requirements, (2) generation time and (3) accuracy. We base our implementation for locating MRFs and RFs on the MRG-Exp and Arima algorithms [37] and use an adapted version of the CFI2TCFI algorithm [13] for producing ϵ-CRFs.

Tuning parameters. The basic parameters that control the generation of the maintained ϵ-CRFs are the support threshold ξ_r for considering a faSet rare and the accuracy-tuning parameter ϵ. Other parameters include the Bloom filter threshold (ξ_0) and the number of employed random walks (as described in Sect. 5.2). In our experiments, as a default, we use a Bloom filter threshold ξ_0 equal to 1 %, except for MOVIES, for which many faSets appear in less than 1 % of the tuples in the dataset. In this case, we use $\xi_0 = 0.01\,\%$ (or around 12 tuples in absolute frequency). Also, we keep the number of random walks fixed (equal to 50 per examined faSet). The values of our tuning parameters are shown in Table 3.

We discretize the numeric values of our real datasets as discussed in Sect. 5.2. In particular, we partition both the production years of movies in the MOVIES dataset and cars in the AUTOS dataset into decades and the price and mileage attributes of the AUTOS dataset into intervals of length equal to 10,000. Throughout our evaluation, we excluded id attributes, since they do not contain information useful in our case.

Effect of ξ_r and ϵ. Table 1 shows the number of generated faSets for our datasets for different values of ξ_r and ϵ. Note that all MRFs are maintained as RFs independently of the number of random walks. As ϵ increases, an ϵ-CRF is allowed to represent faSets with a larger support difference, and thus, the number of maintained faSets decreases. Also, as ξ_r increases, more faSets of the database are considered to be rare, and thus, the size of the maintained information becomes larger. The number of ϵ-CRFs is smaller than the number of RFs, even for small values of ϵ. This is especially evident in the case of the AUTOS dataset, where many faSets have similar frequencies.

Table 2 reports the execution time required for generating faSets. We break down the execution time into three stages: (1) the time required to locate all MRFs, (2) the time required to generate RFs based on the MRFs and (3) the time required to extract the CRFs and the final ϵ-CRFs based on all RFs. We see that the main overhead is induced by the stage of generating the RFs of the database. We can reduce that overhead by decreasing the number of employed random walks. This has a tradeoff with the accuracy of the estimations we receive as we will later see.

To evaluate the accuracy of the estimation of the support of a rare faSet provided by ϵ-CRFs, we randomly construct a number of rare faSets for our datasets. For each dataset, we generate random faSets of length $1, \ldots, \ell$, where ℓ is the largest size for which there exist faSets with count in $(\xi_0, \xi_r]$. Then, we probe our summaries to retrieve estimations for the frequency of 100 such rare faSets for each size. Here, we report results for one synthetic and one real dataset, namely ZIPF-10-2.0 and AUTOS-7. Similar results are obtained

Table 1 Number of generated faSets

ξ_r	# MRFs	# RFs	# CRFs	# ϵ-CRFs					# BF	Pruned
				$\epsilon = 0.1$	$\epsilon = 0.3$	$\epsilon = 0.5$	$\epsilon = 0.7$	$\epsilon = 0.9$		
ZIPF-7-2.0										
5 %	129	1172	1172	1172	1171	680	138	129	1623	27644
10 %	59	1211	1211	1211	1210	707	104	95	1623	28339
20 %	44	1627	1627	1627	1626	942	110	101	1623	35667
ZIPF-10-2.0										
5 %	259	5676	5676	5676	5674	2968	271	259	7753	228213
10 %	106	7065	7065	7065	7063	3778	202	190	7873	267504
20 %	61	10329	10329	10329	10327	5388	202	190	8090	363260
ξ_r	# MRFs	# RFs	# CRFs	#ϵ-CRFs					# BF	Pruned
				$\epsilon = 1.0$	$\epsilon = 2.0$	$\epsilon = 3.0$	$\epsilon = 4.0$	$\epsilon = 5.0$		
ZIPF-7-1.0										
5 %	402	758	758	758	496	403	402	402	3968	32090
10 %	217	927	927	927	491	335	329	263	4108	38665
20 %	84	1032	1032	1032	475	286	280	170	4129	42114
ZIPF-10-1.0										
5 %	838	1895	1895	1895	1075	840	838	838	12174	131843
10 %	430	2377	2377	2377	1076	674	667	537	13014	163887
20 %	135	2772	2772	2772	1064	559	552	327	13266	18715
ξ_r	# MRFs	# RFs	# CRFs	# ϵ-CRFs					# BF	Pruned
				$\epsilon = 0.1$	$\epsilon = 0.3$	$\epsilon = 0.5$	$\epsilon = 0.7$	$\epsilon = 0.9$		
AUTOS-7										
5 %	80	1498	1103	397	243	190	162	133	1438	43569
10 %	74	1882	1404	502	299	238	203	170	1513	57091
20 %	43	2006	1511	531	319	253	216	175	1354	61797
AUTOS-12										
5 %	214	36555	22360	3153	1361	1003	813	661	9225	1740111
MOVIES										
5 %	452	591	556	547	544	543	541	539	68137	2101

for the other datasets as well. Figure 8 shows the average estimation error as a percentage of the actual count of the faSets when varying ϵ and ξ_r (ignore, for now, the dashed lines). We observe that the estimation error remains low even when ϵ increases. For example, it remains under 5 % in all cases for ZIPF-10-2.0. Even though we do not have the complete set of ϵ-CRFs available for our real dataset, because of our random walks approach for producing RFs, the estimation error remains under 15 % for that dataset as well.

Tuning ϵ. Next, we evaluate our heuristic for suggesting ϵ values. Figure 9 depicts the ϵ values and corresponding number of ϵ-CRFs for each of the steps of our tuning algorithm for two of our datasets, namely ZIPF-7-1.0 and AUTOS-7, $\xi_r = 10\,\%$ and various values of the storage limit b. We let our algorithm suggest an ϵ value for each case. The suggested value appears last in the x-axis of each plot. The located ϵ values vary depending on b and the specific dataset. Many times, the storage limit b set by the system administrator may be flexible, i.e., the system administrator may decide to allocate a bit more space, if this results in a significant improvement of ϵ, as is for example the case in Fig. 10a where an increase in b from 330 to 340 leads to decreasing ϵ from 3.932 to 2.364.

Using Bloom filters and varying ξ_0. As previously detailed, Bloom filters can be exploited for fast estimations of faSet frequencies when the number of faSets that appear only a handful of times in the database is large (see, for example, Fig. 11). Table 1 reports the number of faSets inserted into the Bloom filter during the generation of the ϵ-CRFs ("# BF") and the number of faSets that we were able to prune during the generation of RFs because they had a sub-faSet in

Table 2 Execution time (in ms) for generating faSets

ξ_r	MRFs	RFs	CRFs	# ϵ-CRFs				
				$\epsilon = 0.1$	$\epsilon = 0.3$	$\epsilon = 0.5$	$\epsilon = 0.7$	$\epsilon = 0.9$
ZIPF-7-2.0								
5 %	10313	171641	610	657	687	657	750	735
10 %	5219	179031	657	656	719	688	765	782
20 %	1812	268063	1219	1219	1313	1282	1312	1344
ZIPF-10-2.0								
5 %	31203	1421281	16922	17140	16688	16937	17266	17469
10 %	16265	1890844	24265	25046	26281	27781	25844	24328
20 %	5281	2991125	51859	53828	52094	56765	53453	51313
ξ_r	MRFs	RFs	CRFs	# ϵ-CRFs				
				$\epsilon = 0.1$	$\epsilon = 0.3$	$\epsilon = 0.5$	$\epsilon = 0.7$	$\epsilon = 0.9$
ZIPF-7-1.0								
5 %	33484	157250	203	219	219	218	219	219
10 %	8719	197375	297	313	313	344	313	313
20 %	1390	230344	375	407	422	422	453	390
ZIPF-10-1.0								
5 %	98515	680359	1360	1406	1453	1437	1453	2203
10 %	24078	855734	2125	2188	2203	2218	2219	3485
20 %	3703	1081219	3703	3890	3969	3703	3875	4078
ξ_r	MRFs	RFs	CRFs	# ϵ-CRFs				
				$\epsilon = 0.1$	$\epsilon = 0.3$	$\epsilon = 0.5$	$\epsilon = 0.7$	$\epsilon = 0.9$
AUTOS-7								
5 %	22437	1416797	1297	985	1000	985	969	1015
10 %	13984	1977250	2203	1578	1562	1547	1719	1593
20 %	6782	2205453	2453	1797	1891	2125	1937	1891
AUTOS-12								
5 %	98078	54142515	763235	440969	437531	443219	458766	467969
MOVIES								
5 %	149844	15021781	125	125	109	110	109	125

Table 3 Tuning parameters

Parameter	Default value	Range
Estimation factor ϵ	–	0.1–5.0
Rare threshold ξ_r	10 %	5–20%
Bloom filter threshold ξ_0	1 %	0.1–5 %
Random walks per faSet	50	10–50

the Bloom filter ("pruned"). We see that using Bloom filters reduces the cost of generating faSets significantly. Figure 12 shows how the number of the generated MRFs varies as we change the threshold ξ_0 of the Bloom filter for one synthetic and one real dataset and, also, the number of faSets inserted into the Bloom filter during the generation of MRFs. In both cases, we used $\xi_r = 10$ % and varied ξ_0 from 1 to 5 %. We see that, as ξ_0 increases, more faSets are added into the Bloom filter and less MRFs are generated. This has an impact on the following steps of computing RFs, CRFs and ϵ-CRFs, since we avoid storing all possible faSets that are subsumed by some faSet in the Bloom filter. Setting ξ_0 too high, however, excludes many faSets from being considered later by the TPA (Fig. 13).

Effect of random walks. The cost of generating our summaries can be reduced by employing the random walks approach. Figure 13 reports the number of generated ϵ-CRFs for ZIPF-7-2.0 and AUTOS-7 and the corresponding execution time when we vary the number of random walks per faSet. We see that by increasing the number of random walks, we can retrieve more ϵ-CRFs. The generation time

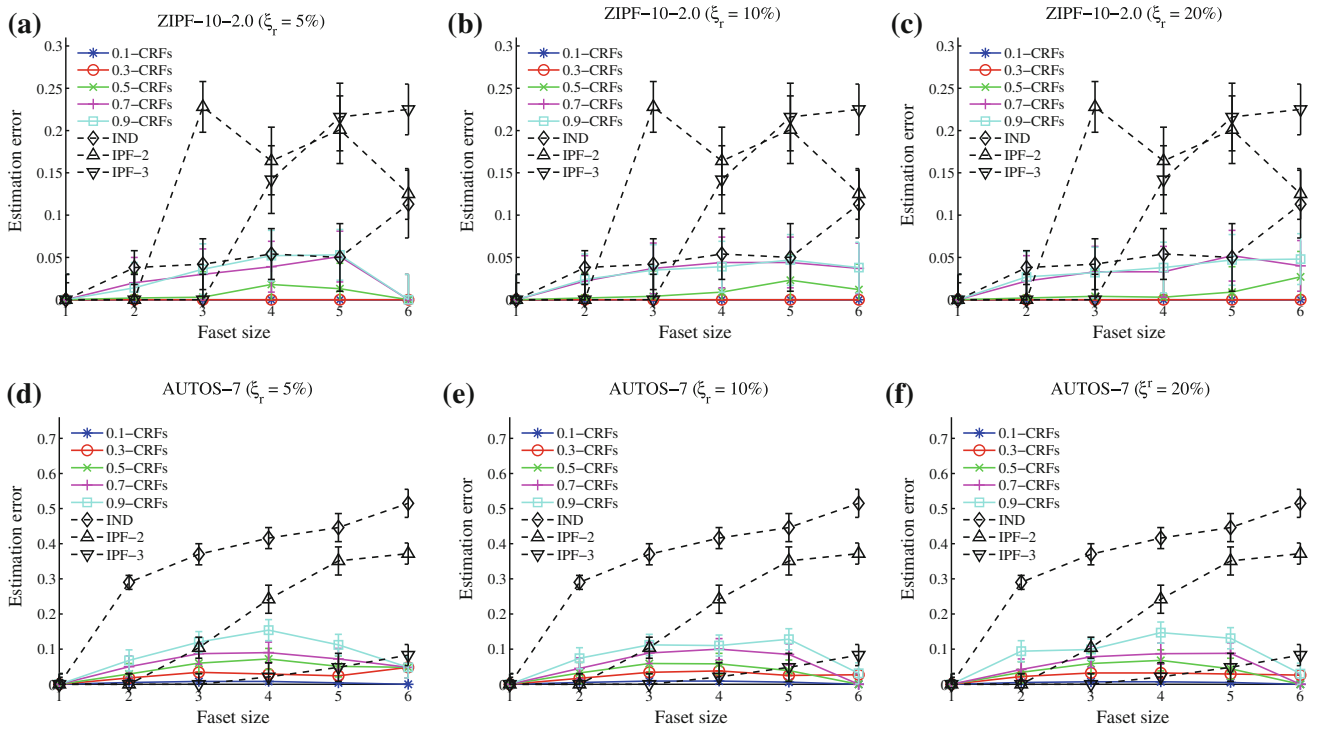

Fig. 8 Estimation error for 100 random rare faSets for different values of ξ_r when varying ϵ. **a** `ZIPF-10-2.0` ($\xi_r = 5\,\%$), **b** `ZIPF-10-2.0` ($\xi_r = 10\,\%$), **c** `ZIPF-10-2.0` ($\xi_r = 20\,\%$), **d** `AUTOS-7` ($\xi_r = 5\,\%$), **e** `AUTOS-7` ($\xi_r = 10\,\%$), **f** `AUTOS-7` ($\xi_r = 20\,\%$)

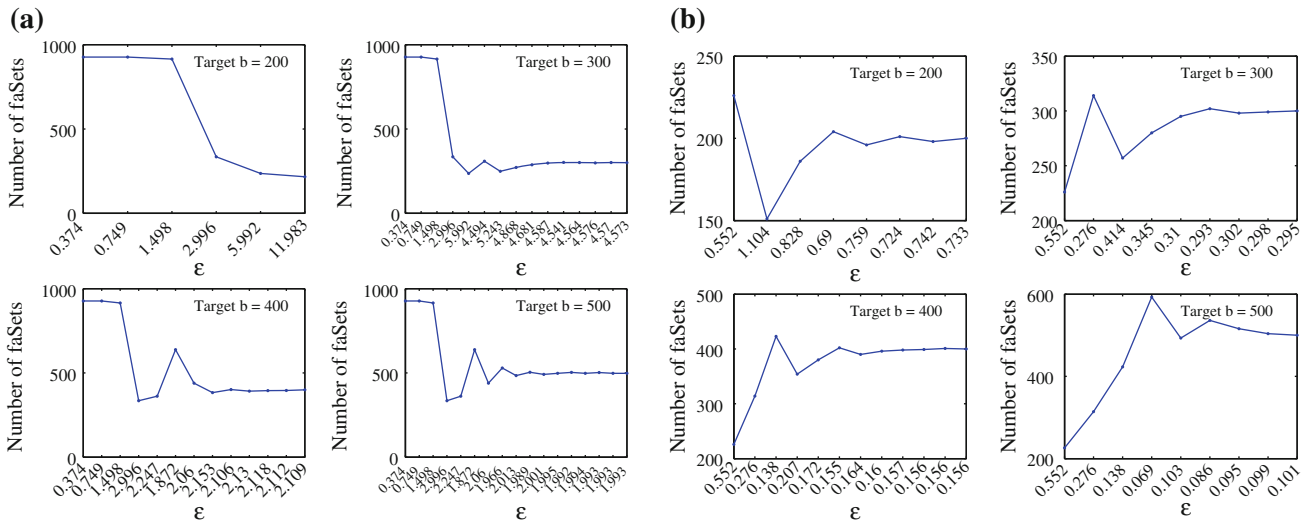

Fig. 9 Automatically suggesting values for ϵ given a storage limit b. **a** `ZIPF-7-1.0` ($\xi_r = 10\,\%$), **b** `AUTOS-7` ($\xi_r = 10\,\%$)

of those ϵ-CRFs is dominated by the time required for the intermediate step of generating the RFs, and thus, ϵ does not affect the execution time considerably.

Next, we evaluate how the estimation accuracy is affected by the number of random walks. We employ our two datasets (`ZIPF-10-2.0` and `AUTOS-7`) and generate ϵ-CRFs for both of them varying the number of random walks used. We use a larger number of random walks for the `AUTOS-7` dataset, since this dataset contains more RFs than the synthetic one. Figure 14 reports the corresponding average estimation error. We see that the estimation error remains low even when fewer random walks are used (Fig. 15).

Exploiting subsumption. We also conduct an experiment to evaluate the performance of our greedy heuristic (Algorithm 1) for exploiting subsumption among faSets of the same

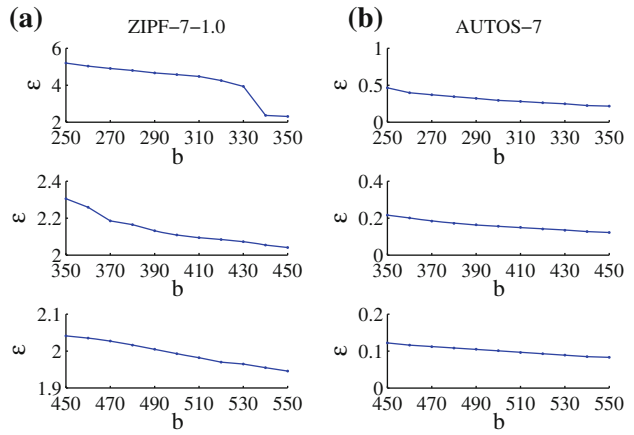

Fig. 10 Suggested ϵ values when varying b. **a** `ZIPF-7-1.0` ($\xi_r = 10\,\%$), **b** `AUTOS-7` ($\xi_r = 10\,\%$)

size. To do this, we randomly generate 10,000 tuples with $|\mathcal{A}| = 1$ taking values uniformly distributed in $[1, v]$ for various values of v. Then, we construct all 1-faSets of the form $(a_i \leq A \leq v)$ where $a_i \in [1, v]$, i.e., there are initially v available faSets. We merge the available faSets using (1) the greedy heuristic (GR) and (2) a random approach where, at each round, we randomly select one of the available faSets and check whether it can ϵ-subsume any other faSets (RA). Figure 16 shows the final number of faSets when varying ϵ, i.e., the size of the corresponding $(1, \epsilon)$-cover sets. We see that merging faSets of the same size can greatly reduce the size of maintained information and that GR produces sets of considerably smaller sizes than those produced by RA. This gain is larger as the number of initially available faSets increases.

6.3.2 Top-k faSet discovery

Next, we compare the baseline and the two-phase algorithms described in Sect. 4. The TPA is slightly modified to take into consideration the special treatment of very rare faSets that have been inserted into the Bloom filter.

To test our algorithms, we generate random queries for the synthetic datasets, while for `AUTOS` and `MOVIES`, we use the example queries shown in Fig. 17. These queries are selected so that their result set includes various combinations of rare and frequent faSets. Figure 15 shows the 1st and 20th highest ranked interestingness score retrieved, i.e., for the TPA, we set $k = 20$, and for the baseline approach, we start with a high ξ_f and gradually decrease it until we get at least 20 results. We see that the TPA is able to retrieve more interesting faSets, mainly due to the first phase where rare faSets of $Res(Q)$ are examined.

We set $k = 20$ and $\xi_r = 5\,\%$ and experimented with various values of ϵ. We saw that ϵ does not affect the interestingness scores of the top-k results considerably. For the above-reported results, ϵ was equal to 0.5. In all cases except for q_3 of the `AUTOS` database, the TPA located k results during phase one, and thus, phase two was never executed. This means that in all cases, there were some faSets present in $Res(Q)$ that were quite rare in the database, and thus, their interestingness was high.

The efficiency of the TPA depends on the size of $Res(Q)$, since in phase one, the tuples of $Res(Q)$ are examined for locating supersets of faSets in the maintained summaries. The TPA was very efficient for result sizes up to a few hundred results, requiring from under a second to around 5 s to run.

6.3.3 Comparison with other methods

We next discuss some alternative approaches for generating database frequency statistics.

Maintaining faSets up to size ℓ. Instead of maintaining a representative subset of rare faSets, we consider maintaining the frequencies of all faSets of up to some specific size ℓ. As an indication for the required space requirements, Table 4 reports the number of faSets up to size 3 for our datasets.

First, let us consider maintaining only 1-faSets and using the independence assumption as described in Sect. 4.1. Figure 8 reports the estimation error when following this alternative approach (denoted "IND"). This approach performs well for the synthetic dataset due to the construction of the dataset, since the values of each attribute are drawn independently from a different zipf distribution. However, this is not the case for the real dataset, where the independence assumption leads to a much larger estimation error than our

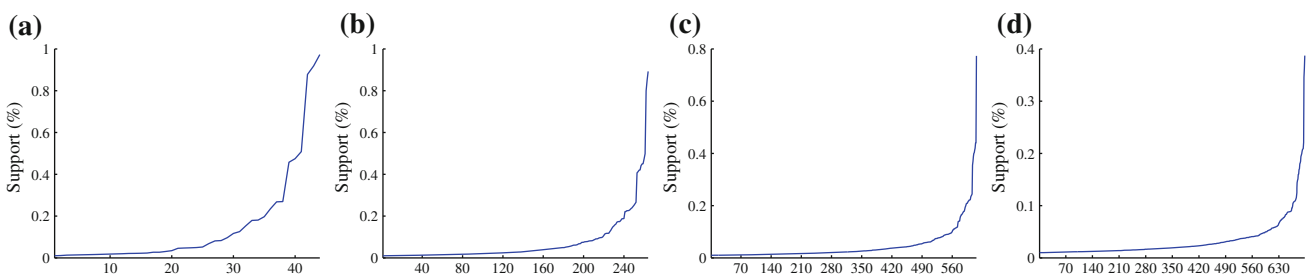

Fig. 11 Support of the faSets for the `AUTOS` dataset (x-axis is the number of faSet ordered by their support, e.g., x=50 means that this is the 50th less frequent faSet. **a** FaSet size 1, **b** FaSet size 2, **c** FaSet size 3, **d** FaSet size 4

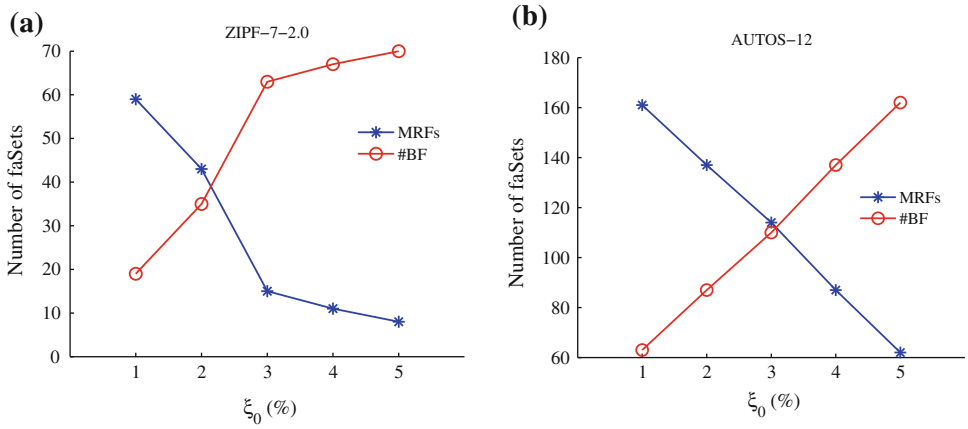

Fig. 12 Number of generated MRFs and number of faSets inserted into the Bloom filter for different values of ξ_0 when $\xi_r = 10\,\%$. **a** `ZIPF-7-2.0`, **b** `AUTOS-12`

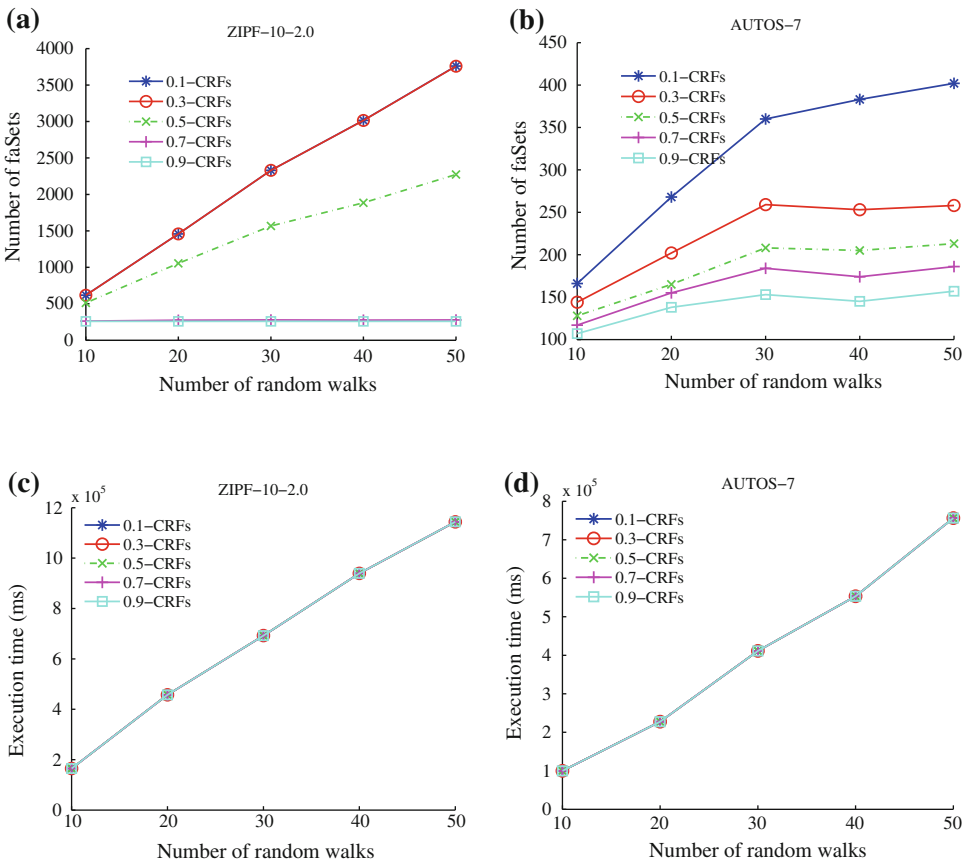

Fig. 13 Number of produced faSets (*top row*) and execution time (*bottom row*) when using different numbers of random walks for generating faSets for $\xi_r = 5\,\%$. **a** `ZIPF-10-2.0`, **b** `AUTOS-7`, **c** `ZIPF-10-2.0`, **d** `AUTOS-7`

approach, even for small values of ϵ. Therefore, this approach cannot be employed in real applications.

Considering that we are willing to afford some extra space to maintain the support of faSets up to size ℓ, $\ell > 1$, a more sophisticated approach is Iterative Proportional Fitting (IPF) [7]. Let $f = \{c_1, \ldots, c_m\}$ be a faSet with size m, $m > \ell$. f can be viewed as the result of a probabilistic experiment: We associate with each selection condition $c_i \in f$ a binary variable. This binary variable denotes whether the corresponding selection condition is satisfied or not. The experiment has $v = 2^m$ possible outcomes. Let p_1 be the probability that the outcome is $(0,\ 0,\ \ldots,\ 0)$, p_2 be the probability that the outcome is $(0,\ 0,\ \ldots,\ 1)$ and so on. That is, p_i is the probability of f being satisfied by exactly the conditions corresponding to the variables equal to 1 as specified by the i^{th} possible outcome, $1 \leq i \leq v$ (see Fig. 18 for an example with $m = 3$ and $\ell = 2$). Having pre-computed the support of faSets up to size ℓ, we have some knowledge (or constraints) for the values of the discrete distribution $\mathbf{p} = (p_1, \ldots, p_v)^T$. First, all p_is for which a faSet f of size m with $m \leq \ell$ is satisfied must sum up to $p(f|\mathcal{D})$, i.e., the pre-computed support. Second, all p_is must sum up to 1. For example, for $\ell = 2$, we have m constraints due to the pre-computed support values of all 1-faSets and $m(m-1)/2$ constraints due to the 2-faSets.

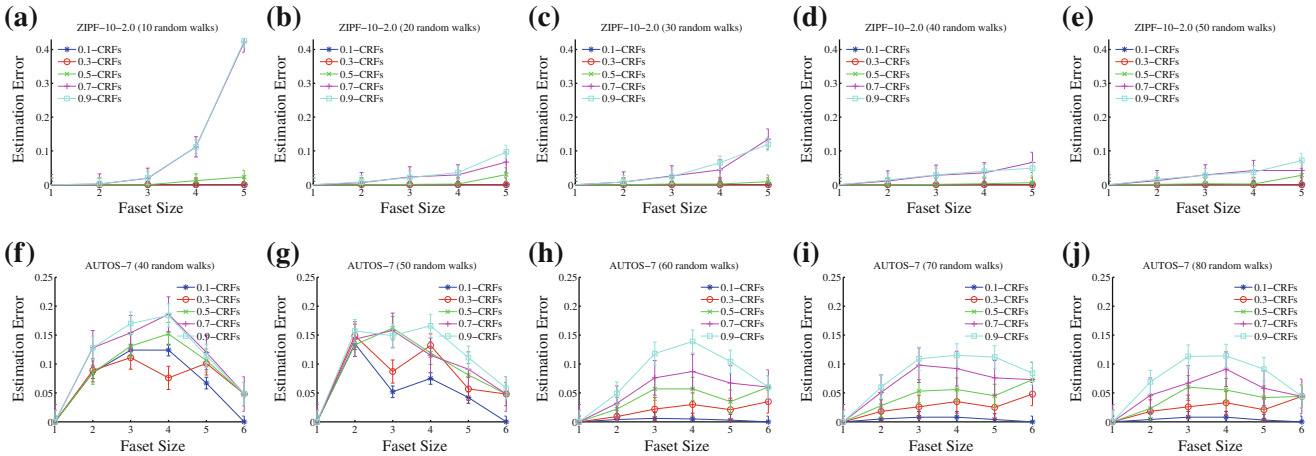

Fig. 14 Estimation error for 100 random rare faSets and $\xi_r = 5\,\%$ for different number of random walks employed during the generation of ϵ-CRFs when varying ϵ. **a** `ZIPF-10-2.0` (10 random walks), **b** `ZIPF-10-2.0` (20 random walks), **c** `ZIPF-10-2.0` (30 random walks), **d** `ZIPF-10-2.0` (40 random walks), **e** `ZIPF-10-2.0` (50 random walks), **f** `AUTOS-7` (40 random walks), **g** `AUTOS-7` (50 random walks), **h** `AUTOS-7` (60 random walks), **i** `AUTOS-7` (70 random walks), **j** `AUTOS-7` (80 random walks)

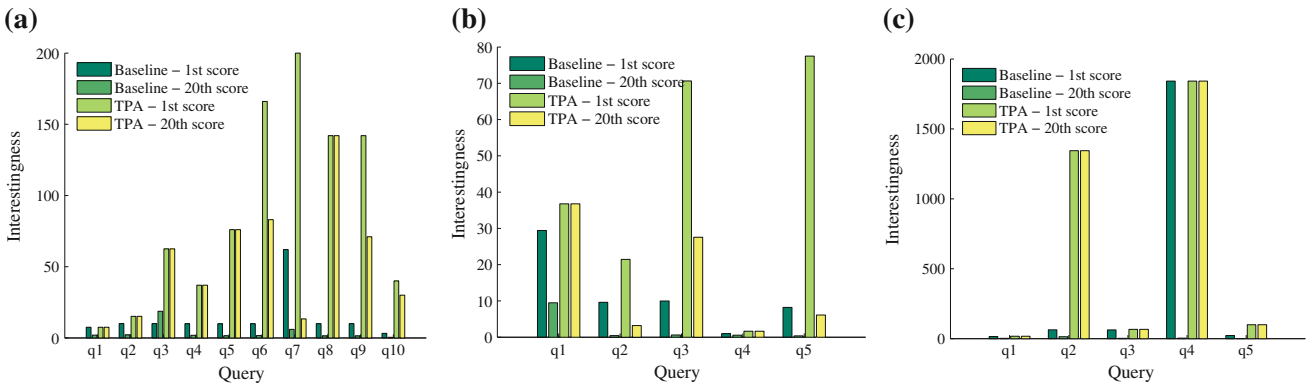

Fig. 15 Interestingness scores of the top 20 most interesting faSets retrieved by the TPA and the baseline approach. **a** `ZIPF-10-2.0`, **b** `AUTOS-12`, **c** `MOVIES`

Therefore, we have $m+m(m-1)/2+1$ constraints in total. However, there are more variables than constraints; therefore, we cannot determine all values of $\mathbf{p}$. IPF is based on the principle of maximum entropy, which states that, since there is no reason to bias the estimated distribution of $\mathbf{p}$ toward any specific form, then the estimation should be as close to the uniform distribution as possible. IPF initializes the elements of $\mathbf{p}$ randomly and then iteratively checks each available constraint and scales by an equal amount the elements of $\mathbf{p}$ participating in the constraint so that the constraint is satisfied. It can be proved that this process converges to the maximum entropy distribution.

The performance of IPF for $\ell = 2$ and $\ell = 3$ is shown in Fig. 8, denoted "IPF-2" and "IPF-3", respectively. We see that for our synthetic dataset, IPF cannot outperform the independence assumption approach. This is not the case for our real dataset, where IPF performs better. Using IPF with $\ell = 2$ results in much higher estimation errors than our ϵ-CRFs approach. Increasing ℓ to 3 improves the performance of IPF. However, this requires maintaining over 6,000 faSets in total for the `AUTOS-7` dataset, while the ϵ-CRFs approach requires up to at most around 500 faSets, depending on the value of ϵ and ξ_r.

In general, our ϵ-CRFs approach provides a tunable method to retrieve frequency estimations of bounded error for rare faSets of any size, without relying on an independence approach. Also, the estimation error does not increase for larger faSets, since we maintain a representative set of not only small faSets, as in the case of IPF, but also larger ones.

Non-derivable faSets. In the case of *frequent* itemsets, an alternative approach for creating compact representations is proposed in [10], where *non-derivable* frequent itemsets are introduced. Non-derivable itemsets can be viewed as an extension of closed frequent itemsets. In particular, a non-derivable frequent itemset I is an itemset whose support cannot be derived based on the supports of its subitemsets. For each sub-itemset, a deduction rule is formed, based on the inclusion/exclusion principle. For example, consider three items a, b and c and let $supp(I)$ (resp. $supp(\overline{I})$) be the number of tuples containing (resp. not containing) I.

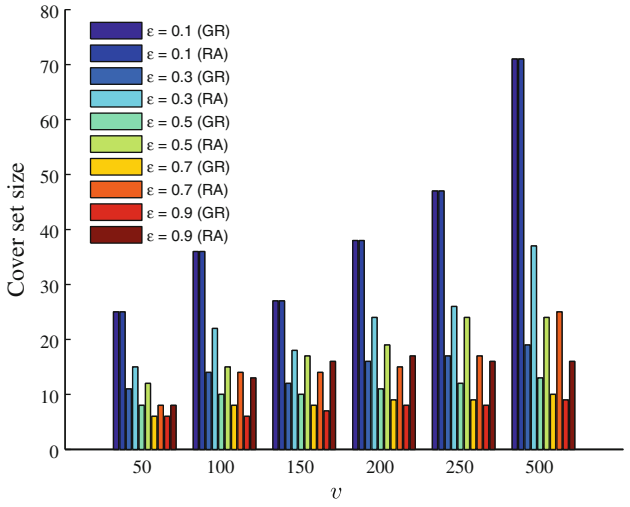

Fig. 16 Size of the produced cover sets by the greedy heuristic (GR) and the random approach (RA) when varying the number of initial faSets v

Table 4 Number of faSets up to size 3

Dataset	# 1-faSets	# 2-faSets	# 3-faSets
ZIPF-7-2.0	70	1901	13681
ZIPF-10-2.0	100	4048	46293
ZIPF-7-1.0	70	2100	31004
ZIPF-10-1.0	100	4500	106497
AUTOS-7	79	1022	4925
AUTOS-12	117	2844	28704
MOVIES	66726	380603	743152

The inclusion/exclusion principle for the itemset $a\overline{bc}$ states that $supp(a\overline{bc}) = supp(a) - supp(ab) - supp(ac) + supp(abc)$. Since $supp(a\overline{bc})$ must be greater than or equal to zero, we can deduce that $supp(abc) \geq supp(ab) + supp(ac) - supp(a)$. Generally, for every subset X of I, it is shown that:

$$supp(I) \leq \sum_{X \subseteq J \subset I} (-1)^{|I \setminus J|+1} supp(J), \quad \text{if} \quad |I \setminus X| \text{ is odd}$$

$$supp(I) \geq \sum_{X \subseteq J \subset I} (-1)^{|I \setminus J|+1} supp(J), \quad \text{if} \quad |I \setminus X| \text{ is even}$$

Therefore, for each itemset I, there are a number of rules providing upper and lower bounds for I. Let u_I and l_I be these bounds, respectively. If $u_I = l_I$, then we can deduce that $supp(I) = u_I$ and I is a derivable itemset. The monotonicity property holds for derivable itemsets, i.e., if I is derivable, then every superset of I is derivable as well. Thus, an Apriori-like algorithm is employed to generate all non-derivable frequent itemsets.

There are two non-trivial extensions that need to be addressed for applying non-derivability in our case. First, a method is required for generating non-derivable rare faSets, instead of frequent ones. Second, frequency bounding must be extended to allow approximations of the frequencies of the various faSets.

q_1: **select** make, name
from autos
where navigation_system = 'Yes';

q_2: **select** make, air_condition, alarm
from autos
where state = 'FL';

q_3: **select** make, name, sunroof
from autos
where state = 'MD' **and** make = 'Lexus';

q_4: **select** make, state, spoiler
from autos
where air_condition = 'Yes' **and** power_steering = 'Yes';

q_5: **select** make, state, side_air_bag
from autos
where child_safety = 'Yes' **and** cruise_control = 'Yes';

(a) AUTOS.

q_1: **select** D.name, G.genre, M.year
from C, M, D, M2D, G
where *join* **and** C.country='France';

q_2: **select** C.country, G.genre, M.year
from C, M, D, M2D, G
where *join* **and** D.name='Coppola, Francis Ford';

q_3: **select** C.country, M.year
from C, M, D, M2D, G
where *join* **and** M2D.notes='Uncredited';

q_4: **select** D.name, G.genre
from C, M, D, M2D, G
where *join* **and** D.name='Coppola, Francis Ford' **and** M2D.notes='Uncredited';

q_5: **select** D.name, C.country
from D, M2D, M, C
where *join* **and** M.year=2000;

(b) MOVIES.

Fig. 17 Dataset queries used for evaluating TPA and the baseline approach. **a** AUTOS, **b** MOVIES.

D.name = "M. Scorsese"	*M.year = 2010*	*G.genre = "Action"*	probability
0	0	0	p_1
0	0	1	p_2
0	1	0	p_3
0	1	1	p_4
1	0	0	p_5
1	0	1	p_6
1	1	0	p_7
1	1	1	p_8

$p_5 + p_6 + p_7 + p_8 = p(\{D.name = \text{“M. Scorsese”}\})$
$p_3 + p_4 + p_7 + p_8 = p(\{M.year = 2010\})$
$p_2 + p_4 + p_6 + p_8 = p(\{G.genre = \text{“Action”}\})$
$p_7 + p_8 = p(\{D.name = \text{“M. Scorsese”}, M.year = 2010\})$
$p_6 + p_8 = p(\{D.name = \text{“M. Scorsese”}, G.genre = \text{“Action”}\})$
$p_4 + p_8 = p(\{M.year = 2010, G.genre = \text{“Action”}\})$
$p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 = 1$

Fig. 18 Example of IPF constraints for $\ell = 2$ when estimating the support of the faSet {*D.name*="*M. Scorsese*", *M.year*="*2010*", *G.genre*="*Action*"}

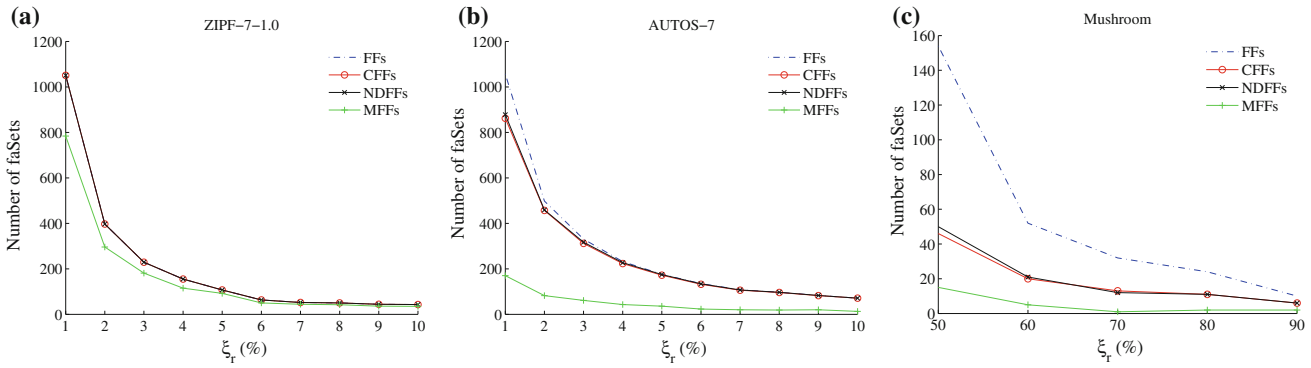

Fig. 19 Frequent, closed frequent, non-derivable frequent and minimal frequent faSets for various datasets. **a** `ZIPF-7-1.0`, **b** `AUTOS-7`, **c** `Mushroom`

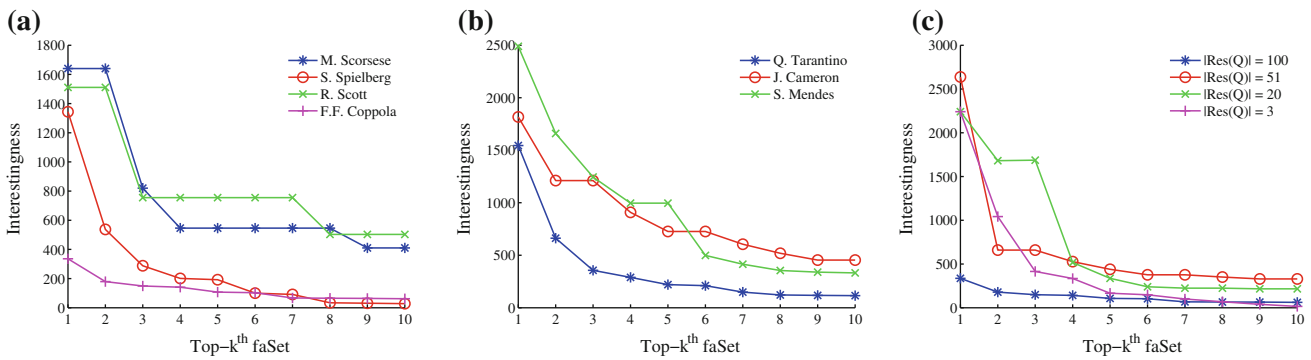

Fig. 20 Interestingness of the top-10 faSets for queries with different number of results. **a** Different queries ($|Res(Q)| \approx 100$), **b** Different queries ($|Res(Q)| \approx 30$), **c** Different ($|Res(Q)|$)

The first issue seems to not be easily solvable. When constructing deduction rules for a faSet I, it is assumed that the frequencies of all its sub-faSets are either stored or can be derived from the frequencies of the stored faSets. This is not the case for a rare faSet I, since many of the sub-faSets of I may be frequent, and thus, their frequency may not be known. We considered following a reverse approach of forming deduction rules based on super-faSets. However, we saw that only lower bounds can be derived from such rules. The second issue could be addressed by relaxing the notion of derivability and allow the upper and lower bounds of a faSet to differ by a factor δ. However, such an extension is not clear, even for frequent faSets, since it is not clear how the estimation error is bounded. The reason for this is that bounds are computed based on multiple deduction rules where many different sub-itemsets participate.

Nevertheless, to get some intuition about the prospects of employing non-derivable faSets, we conducted an experiment for frequent faSets. Figure 19 reports the number of FFs, MFFs, CFFs and non-derivable frequent faSets (NDFFs) for our `ZIPF-7-1.0` and `AUTOS-7` datasets. The number of CFFs and NDFFs is almost identical to that of FFs for our datasets, even for small values of ξ_f down to 1 %, where almost all faSets are considered frequent. This is due to the fact that most faSets in our datasets have distinct frequencies, and thus, the upper and lower bounds derived for the various itemsets are not equal. Figure 19 also reports results for `Mushroom` [2], a dataset widely used in the literature of frequent itemset mining which does not have the same property. Employing CFFs and NDFFs performs better for this dataset. However, we see that the numbers of CFFs and NDFFs are comparable.

6.3.4 Impact of result size

Next, we study the impact of the query result size on the usefulness of our method. In general, the interestingness of a faSet does not depend on the result size per se but rather on the specific query. To illustrate this, we report the interestingness score of the top-10 faSets of different queries with roughly the same result size, in particular of queries retrieving the country, year and genre of movies by a number of different directors that have directed around 100 (Fig. 20a) and 30 (Fig. 20b) movies each. We see that, even though these queries have the same result size, the interestingness of their faSets depends on the specific selection conditions, i.e., director, of the query.

We also consider queries about movies of the same director, namely F.F. Coppola, each retrieving a different number of results (Fig. 20c). These queries produce many com-

q_1: **select** G.genre, M.year **from** G, M, M2D, D **where** *join* **and** D.name = '...'
q_2: **select** M.year, G.genre, D.name **from** G, M, D, M2D, C **where** *join* **and** C.country = '...'
q_3: **select** C.country, D.name **from** G, M, D, M2D, C **where** *join* **and** G.genre = '...' **and** C.country = '...'
q_4: **select** G.genre **from** G, M, A, M2A **where** *join* **and** A.name = '...' **and** M.year = '...'
q_5: **select** M.year, C.country **from** G, M, C **where** *join* **and** G.genre = '...'

Fig. 21 Query templates for the MOVIES database used for the user evaluation

mon faSets which (especially the top 1-3 ones) get a higher interestingness scores for smaller result sizes, since in this case, their support in $|Res(Q)|$ is larger. However, the relative ranking of these common faSets is the same in all queries. Thus, the output of our approach is not affected by the result size of the query (Fig. 17).

6.4 User evaluation

To evaluate the usefulness of YMALDB and its various aspects, we conducted an empirical evaluation using the MOVIES dataset, with 20 people having a moderate interest in movies, 12 of which were computer science graduate students, while the rest of them had no related background. Although this may be considered a relatively small group of users, it provides an indication of the potential impact of our approach.

Users were first introduced to the system and were given some time to familiarize themselves with the interface. Then, each user was allowed to submit a number of queries to the system. A set of template queries was available (Fig. 21) which users could adjust by filling in their preferred directors, actors, genres and so on. Users could also submit non-template queries. All users started by submitting template queries. As they became more comfortable with the system, many of our computer science users started experimenting with their own (non-template) queries. The result size of the various queries was between 20 and 6,700 tuples. User feedback seemed to be independent of the size of the retrieved result set.

We evaluated the effectiveness of the system in two ways: first, by asking users to explicitly comment on the usefulness of the various aspects of the system and, second, by monitoring their interactions with the system. More specifically, users were asked to evaluate the following aspects: (1) the presentation of interesting faSets as an intermediate step before presenting recommendations, (2) the quality of the recommendations, (3) the usefulness of explanations, (4) the usefulness of attribute expansion and (5) the depth of exploration which can also be seen as an indication of the user engagement with the system. Concerning system interaction, we monitored: (1) how many template and non-template queries the users submitted, (2) how many and which interesting faSets the users clicked on for each query, (3) how many and which recommendations users were interested, and (4) how many exploration steps the users followed, i.e., how many exploratory queries initiated at the originally submitted user query were submitted. Table 5 summarizes our findings. We also report the variation in these values. These variations are relatively small and seems to be attributed to behavioral habits whose analysis is beyond the scope of this paper. We present some related comments along with the results.

Interesting faSets. All users preferred being presented first with interesting faSets instead of being presented directly with recommendations. Almost all users preferred seeing interesting faSets grouped in categories according to the attributes they contain. They felt that this made it easier for them to focus on the attributes that they found to be more interesting, which were different for each submitted query. In particular, one user found this grouping interesting in itself, in the sense that it provided a summary of the most important aspects of the result of the original query. Only two computer science users stated that, even though they generally preferred being presented with grouped faSets, they also liked being presented with the top-5 most interesting faSets independently of their categories. All of our non–computer science users found this global top-k confusing and preferred seeing only faSets grouped into categories. For this reason, we decided to let users enable or disable this feature. Also, most users, independently of their background, were more interested in categories corresponding to large faSets because they felt that these faSets were more informative.

Our monitoring concerning which faSets users eventually clicked on showed that there were two types of users, those that clicked on the first one or two faSets from each category and those that chose one or two categories that seemed the most interesting to them and then proceeded with clicking all faSets in these categories, often using the "More" button to retrieve more faSets in these categories. Around 75 % of our users belonged to the former type.

Recommendations. Concerning recommendations, we observed that the exploratory queries that users decided to proceed with depended on the specific attributes for which recommendations were being made. For example, when recommending movie years or genres, around 80 % of the users decided to click on the first couple of available recommendations. However, when recommending actors or directors, the same users clicked on the names they were more familiar with. This supports our decision to rank our recommendations based on the popularity of values in the dataset.

Table 5 Summary of the results of the user study

	User comments	Clicks
Query submission	Computer science students preferred non-template queries/others felt more comfortable with template queries	5 template queries for all users
		2–3 non-template queries on average in addition for computer science users (min = 1, max = 5)
Interesting faSets	Liked the attribute grouping preferred faSets with more attributes	75 % 1–2 faSets of all groups (breadth exploration) (min = 1, max = 4) 25 % all faSets of 1–2 groups (depth exploration) (min = 1, max = 3)
Recommendations	Choice depends on attributes	80 % on the first 1–2 recommendations (genre, year, etc.) and on all recommendations known to them (actors, directors) 20 % on many recommendations (up to 8)
Explanations	Brief	–
	Optional	
Attribute expansion	80 % liked it	Over 90 % clicked first on expanded faSets
	20 % found it arbitrary	
Exploration depth	–	70 % a "close" neighborhood of the original query (1-2 steps) 30 % navigate away (6–7 steps)

Explanations. Contrary to what we expected, user feelings toward using explanations were mixed. Generally, the more users became familiarized with using the system, the less useful they found explanations. Explanations were better received by our non-computer science users, since our computer science users were more interested in understanding how our ranking algorithm works rather than reading the explanations. Nevertheless, all users agreed that explanations should be brief, as they felt that detailed explanations would only clutter the page. Following user feedback, we added an option to allow users to turn explanations off.

Attribute expansions. Around 70 % of our computer science users and all others found query expansion very useful, as they received more recommendations. This behavior seems to be linked with the fact that users preferred seeing larger interesting faSets, since more such faSets appear when expanding queries. Some of our users felt that query expansion was able to retrieve more "hidden" information from the database, which was something they liked.

Exploration depth. Finally, concerning the amount of exploration steps followed by the users, again, there were two types of users. Almost 70 % of the users decided to explore a close "neighborhood" around their original query (1–2 exploration steps), by following a recommendation and then navigating back to the previous page to select a different recommendation for their original query. The remaining users, after following a recommendation and seeing the results and the new interesting recommendations of the corresponding exploring query, would most often find something interesting in the new recommendations and navigate further away from their original query (6–7 exploration steps on average), most often never returning back to the initial page from which their exploration originated. As an example of such an exploration, upon asking for thriller movies in 2006, one of our users followed an interesting faSet about Germany and a consequent recommendation about war movies in 2009. The interesting faSets of the corresponding exploratory query included the countries Serbia and Bosnia and Herzegovina as well as Pantelis Voulgaris, which is a director of civil war movies in Greece.

7 Related work

In this paper, we have proposed a novel database exploration model based on exploring the results of user queries. Another exploration technique is *faceted search* (e.g., [16,20,30]), where results of a query are classified into different multiple categories, or facets, and the user refines these results by selecting one or more facet condition. Our approach is different in that we do not tackle refinement. Our goal is to identify faSets, possibly expand them and then use them to discover other interesting results that are not part of the original query results.

There is also some relation with *query reformulation*. In this case, a query is relaxed or restricted when the number of results of the original query is too few or too many, respectively, using term rewriting or query expansion to increase the recall and precision of the original query (e.g., [32]). Again, our aim is not to increase or decrease the number of retrieved query results but to locate and present interesting results that, although not part of the original query, are highly related to it. Besides reformulating the query, another common method of addressing the too many answers problem is

ranking the results of a query and presenting only the top-k most highly ranked ones to the user. This line of research is extensive; the work most related to ours is research based on automatically ranking the results [5,12]. Besides addressing a different problem, our approach is also different in that the granularity of ranking in our approach is in the level of faSets as opposed to whole tuples. We also propose a novel method for frequency estimation that does not rely on an independence assumption.

Yet another method of exploring results relies on *why queries* that consider the presence of unexpected tuples in the result and *why not* queries that consider the absence of expected tuples in the result. For example, ConQueR [41] proposes posing follow-up queries for *why not* by relaxing the original query. In our approach, we find interesting faSets in the result based on their frequency and other faSets highly correlated with them. Another related problem is constructing a query whose execution will yield results equivalent to a given result set [33,42]. Our work differs in that we do not aim at constructing queries but rather guiding the users toward related items in the database that they may be unaware of.

Other approaches toward making database queries more user-friendly include query auto-completion (e.g., [21]) and free-form queries (e.g., [34]). Khoussainova et al. [21] consider the auto-completion of SQL user queries while they are being submitted to the database. In our work, we consider the expansion of user queries to retrieve more interesting information from the database. The focus of our work is not on assisting users in query formulation but rather on exploring query results for locating interesting pieces of information. [34] considers exploiting database relations that are not part of user queries to locate information that may be useful to the users. The focus of this work, however, is on allowing users to submit free-form, or unstructured, queries and provide answers that are close to a natural language representation.

In some respect, exploratory queries may be seen as recommendations. Traditional recommendation methods are generally categorized into *content-based* that recommend items similar to those the user has preferred in the past (e.g. [26,29]) and *collaborative* that recommend items that similar users have liked in the past (e.g. [9,22]). Adomavicius and Tuzhilin [4] provide a comprehensive survey of the current generation of recommendation systems. Several extensions have been proposed, such as extending the typical recommenders beyond the two dimensions of users and items to include further contextual information [28]. Here, we do not exploit such information but rather rely solely on the query result and database frequency statistics.

Extending database queries with recommendations has been suggested in some recent works, namely [24] and [6,11]. Koutrika et al. [24] propose a general framework and a related engine for the declarative specification of the recommendation process. Our recommendations here are of a very specific form. Recommendations in [6,11] have the form of queries and are based on the relations they involve and the similarity of their structure to that of the original user query. Given past behavor of other users, the goal is to predict which tuples in the database the user is interested in and recommend suitable queries to retrieve them. Those recommendations are based on the past behavior of similar users, whereas we consider only the content of the database and the query result.

A somewhat related problem is finding interesting or *exceptional* cells in an OLAP cube [31]. These are cells whose actual value differs substantially from the anticipated one. The anticipated value for a cell is estimated based on the values of its adjacent cells at all levels of group-bys. The techniques used in that area are different though, and no additional items are presented to the users. Giacometti et al. [17] consider recommending to the users of OLAP cubes queries that may lead to the discovery of useful information. This is a form of database exploration. However, such recommendations are computed based on the analysis of former querying sessions by other users. Here, we do not exploit any history or query logs but, instead, we use only the result of the user query and database information.

Finally, note that we base the computation of interestingness for our results on the interestingness score. There is a large number of possible alternatives none of which is consistently better than the others in all application domains (see [38] for a collection of such measures). In this paper, we use an intuitive definition of interestingness that depends on the relative frequency of each piece of information in the query result and the database. Nevertheless, our exploration framework could be employed along with some different interestingness measure as well by adapting the estimation of interestingness scores accordingly.

This paper is an extended version of [14] including generalized faSets with range conditions, a prototype system and a user evaluation. Some of our initial ideas on this line of research appeared in [36].

8 Conclusions and future work

In this paper, we presented a novel database exploration framework based on presenting to the users additional items which may be of interest to them although not part of the results of their original query. The computation of such results is based on identifying the most interesting sets of (attribute, value) pairs, or faSets, that appear in the result of the original user query. The computation of interestingness is based on the frequency of the faSet in the user query result and in the database instance. Besides proposing a novel mode of exploration, other contributions of this work include a frequency estimation method based on storing an ϵ-tolerance

CRFs representation and a two-phase algorithm for computing the top-k most interesting faSets.

There are many directions for future work. One such direction is to explore those faSets that appear in the result set less frequently than expected, that is, the faSets that have the smallest interestingness value. Such faSets seem to be the ones most loosely correlated with the query and they could be used to construct exploratory queries of a different nature. Another interesting line for future research is to apply our faSet-based approach in the case in which a history of previous database queries and results is available. In this case, the definition of interestingness should be extended to take into consideration the frequency of faSets in the history of results.

Acknowledgments The research of the first author has been co-financed by the European Union (ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the NSRF - Research Funding Program: Heracleitus II. The research of the second author has been co-financed by the European Union (ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the NSRF - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund EICOS project.

References

1. IMDb. http://www.imdb.com
2. Mushroom. http://archive.ics.uci.edu/ml/datasets/Mushroom
3. Yahoo!Auto. http://autos.yahoo.com
4. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Trans. Knowl. Data Eng. **17**(6), 734–749 (2005)
5. Agrawal, S., Chaudhuri, S., Das, G., Gionis, A.: Automated ranking of database query results. In: CIDR (2003)
6. Akbarnejad, J., Chatzopoulou, G., Eirinaki, M., Koshy, S., Mittal, S., On, D., Polyzotis, N., Varman, J.S.V.: Sql querie recommendations. PVLDB **3**(2), 1597–1600 (2010)
7. Bishop, Y.M., Fienberg, S.E., Holland, P.W.: Discrete Multivariate Analysis: Theory and Practice. Springer, New York (2007)
8. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
9. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: UAI (1998)
10. Calders, T., Goethals, B.: Non-derivable itemset mining. Data Min. Knowl. Discov. **14**(1), 171–206 (2007)
11. Chatzopoulou, G., Eirinaki, M., Polyzotis, N.: Query recommendations for interactive database exploration. In: SSDBM (2009)
12. Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic information retrieval approach for ranking of database query results. ACM Trans. Database Syst. **31**(3), 1134–1168 (2006)
13. Cheng, J., Ke, Y., Ng, W.: Delta-tolerance closed frequent itemsets. In: ICDM (2006)
14. Drosou, M., Pitoura, E.: Redrive: result-driven database exploration through recommendations. In: CIKM (2011)
15. Garcia-Molina, H., Koutrika, G., Parameswaran, A.G.: Information seeking: convergence of search, recommendations, and advertising. Commun. ACM **54**(11), 121–130 (2011)
16. Garg, S., Ramamritham, K., Chakrabarti, S.: Web-cam: monitoring the dynamic web to respond to continual queries. In: SIGMOD (2004)
17. Giacometti, A., Marcel, P., Negre, E., Soulet, A.: Query recommendations for olap discovery-driven analysis. IJDWM **7**(2), 1–25 (2011)
18. Gunopulos, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., Sharm, R.S.: Discovering all most specific sentences. ACM Trans. Database Syst. **28**(2), 140–174 (2003)
19. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco (2000)
20. Kashyap, A., Hristidis, V., Petropoulos, M.: Facetor: cost-driven exploration of faceted query results. In: CIKM (2010)
21. Khoussainova, N., Kwon, Y., Balazinska, M., Suciu, D.: Snipsuggest: context-aware autocompletion for sql. PVLDB **4**(1), 22–33 (2010)
22. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J.: Grouplens: applying collaborative filtering to usenet news. Commun. ACM **40**(3), 77–87 (1997)
23. Koudas, N., Li, C., Tung, A.K.H., Vernica, R.: Relaxing join and selection queries. In: VLDB (2006)
24. Koutrika, G., Bercovitz, B., Garcia-Molina, H.: Flexrecs: expressing and combining flexible recommendations. In: SIGMOD (2009)
25. Lee, Y.K., Kim, W.Y., Cai, Y.D., Han, J.: Comine: efficient mining of correlated patterns. In: ICDM (2003)
26. Mooney, R.J., Roy, L.: Content-based book recommending using learning for text categorization. CoRR cs.DL/9902011 (1999)
27. Omiecinski, E.: Alternative interest measures for mining associations in databases. IEEE Trans. Knowl. Data Eng. **15**(1), 57–69 (2003)
28. Palmisano, C., Tuzhilin, A., Gorgoglione, M.: Using context to improve predictive modeling of customers in personalization applications. IEEE Trans. Knowl. Data Eng. **20**(11), 1535–1549 (2008)
29. Pazzani, M.J., Billsus, D.: Learning and revising user profiles: the identification of interesting web sites. Mach. Learn. **27**(3), 313–331 (1997)
30. Roy, S.B., Wang, H., Das, G., Nambiar, U., Mohania, M.K.: Minimum-effort driven dynamic faceted search in structured databases. In: CIKM (2008)
31. Sarawagi, S., Agrawal, R., Megiddo, N.: Discovery-driven exploration of olap data cubes. In: EDBT (1998)
32. Sarkas, N., Bansal, N., Das, G., Koudas, N.: Measure-driven keyword-query expansion. PVLDB **2**(1), 121–132 (2009)
33. Sarma, A.D., Parameswaran, A.G., Garcia-Molina, H., Widom, J.: Synthesizing view definitions from data. In: ICDT (2010)
34. Simitsis, A., Koutrika, G., Ioannidis, Y.E.: Précis: from unstructured keywords as queries to structured databases as answers. VLDB J. **17**(1), 117–149 (2008)
35. Srikant, R., Agrawal, R.: Mining quantitative association rules in large relational tables. In: SIGMOD (1996)
36. Stefanidis, K., Drosou, M., Pitoura, E.: "you may also like" results in relational databases. In: PersDB (2009)
37. Szathmary, L., Napoli, A., Valtchev, P.: Towards rare itemset mining. In: ICTAI (1) (2007)
38. Tan, P.N., Kumar, V., Srivastava, J.: Selecting the right interestingness measure for association patterns. In: KDD (2002)
39. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison Wesley, Boston (2005)
40. Tintarev, N., Masthoff, J.: Designing and evaluating explanations for recommender systems. In: Recommender Systems Handbook (2011)
41. Tran, Q.T., Chan, C.Y.: How to conquer why-not questions. In: SIGMOD (2010)
42. Tran, Q.T., Chan, C.Y., Parthasarathy, S.: Query by output. In: SIGMOD (2009)