open proceedings

TimeReach: Historical Reachability Queries on Evolving Graphs

Konstantinos Semertzidis, Kostas Lillis, Evaggelia Pitoura
Computer Science and Engineering Department
University of Ioannina, Greece
{ksemer,klillis,pitoura}@cs.uoi.gr

ABSTRACT

Since most graphs evolve over time, it is useful to be able to query their history. We consider historical reachability queries that ask for the existence of a path in some time interval in the past, either in the whole duration of the interval (conjunctive queries), or in at least one time instant in the interval (disjunctive queries). We study both alternatives of storing the full transitive closure of the evolving graph and of performing an online traversal. Then, we propose an appropriate reachability index, termed TimeReach index, that exploits the fact that most real-world graphs contain large strongly connected components. Finally, we present an experimental evaluation of all approaches, for different graph sizes, historical query types and time granularities.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems query processing

General Terms

Algorithms, Measurement, Performance

Keywords

Evolving Graphs, Historical Queries, Reachability

1. INTRODUCTION

In recent years, increasing amounts of graph structured data are being made available from a variety of sources, such as social, citation, computer and hyperlink networks. Almost all such real-world networks evolve over time, as nodes and edges are added or deleted. Analysis of their evolution finds a large spectrum of applications, ranging from social network marketing, to virus propagation and digital forensics.

In this paper, we assume that we are given an evolving set of graph snapshots corresponding to the state of the graph at different time instants. We address the problem of efficiently answering queries that involve such snapshots. In particular, we focus on a basic query type, namely reachability queries, that ask whether a node u was reachable from another node v during specific time intervals in the past. We call such queries *historical reachability queries*.

Although, there has been considerable interest in processing graph data, through a variety of graph queries including reachability, distance and pattern-based ones, querying the graph history is much less studied. The only other two approaches to building indexes for processing historical graph queries that we are aware of consider historical shortest-path queries [9, 2]. Specifically, the authors of [9] propose a method based on ordering nodes or edges pertinent to shortest path computation, while the dynamic index construction proposed in [2] does not support node or edge deletions.

All other work on historical queries focuses mainly on efficiently storing and retrieving the graph snapshots required for processing each query [14, 13, 21, 17]. In particular, in [14], a combination of graph deltas and selected materialized snapshots are explored, while in [13], the focus is on storing, sharing and processing deltas. In [21], temporally close snapshots are clustered, one representative per cluster is selected and used for an initial evaluation of the query. Finally, in [17], the placement and replication of snapshots in a distributed setting is studied. Instead, in this paper, we address the problem of building indexes for answering historical reachability queries.

Reachability queries on static graphs have been extensively studied. Research in this area follows two general directions through efficiently storing the transitive closure and speeding-up online traversal. With regards to transitive closure, various approaches have been proposed including the chain method [10, 5], methods exploring spanning trees, bit-vector compression [26] and interval [1, 28, 12], and hop [7, 22, 6] labeling. In the case of online traversal, often interval labeling [4, 25, 30] is used to prune the search space. There has also been some work on incrementally maintaining the reachability indexes in case of evolving graphs [1, 3, 23, 31], however, reachability still considers a single snapshot, i.e., the current version of the graph.

In this paper, we explore a compact representation of graph snapshots, called *version graph*, where each node and edge is annotated with the set of time intervals during which the corresponding node and edge existed in the evolving graph. We call such sets *lifespans* and seek for their minimum representation through using non-overlapping and non-continuous intervals. We also introduce a set of basic operations for efficiently manipulating lifespans of paths.

10.5441/002/edbt.2015.12

For processing historical reachability queries, we start by revisiting the basic transitive closure and online traversal approaches. For the transitive closure, we compute a minimum representation of reachability information for each pair of nodes. For the online traversal, we propose a novel interval-based traversal of the version graph along with a number of pruning steps. Furthermore, to avoid the cost and space overheads associated with precomputing the transitive closure and improving the processing cost of the online traversal, we propose a new approach, termed *TimeReach*.

TimeReach exploits the fact that most graphs consist of strongly connected components (SCCs) [20, 15]. Thus, instead of maintaining reachability information for pairs of nodes, we maintain *posting lists* with information about node membership in SCCs. We minimize the size of posting lists through an appropriate assignment of identifiers to SCCs. We show that the problem of the optimal assignment of identifiers to SCCs is equivalent to the maximum bipartite matching problem among SCCs in consequent graph snapshots. Along with postings, we maintain a condensed version graph which corresponds to the version graph of the SCCs evolution. To improve the performance of answering historical queries, we also introduce an interval-2hop approach based on pruned landmark labeling [2, 29] on the condensed version graph.

We have extensively evaluated our approach with three real social network datasets. Our experimental results show that TimeReach is space efficient, in particular for graphs consisting of large SCCs as is the case of social networks. Its incremental construction is fast; indexing a new snapshot graph takes just a few seconds. Finally, processing historical queries using TimeReach is orders of magnitude faster than the online traversal of the version graph.

The rest of this paper is structured as follows. In Section 2, we present related work, while in Section 3, we formally define historical reachability queries. In Section 4, we introduce the version graph and operations on lifespans and present the two baseline approaches, namely, the transitive closure and online traversal. In Section 5, we introduce the TimeReach Index approach, while in Section 6, we present experimental results. Section 7 concludes the paper.

2. RELATED WORK

Although, graph data management has been the focus of much current research, work in processing historical queries is rather limited. The main focus of research on evolving graphs has been on efficiently storing and retrieving graph snapshots. In this paper, our focus in on indexing for processing queries. To this end, we assume a compact representation of the sequence of graph snapshots in the form of a version graph. Alternatively, one can store just some subset of the graph snapshots in the sequence along with appropriate deltas, such that, any other snapshot can be reconstructed by applying the deltas on the selected snapshots [14, 13]. Various optimizations for reducing the storage and snapshot re-construction overheads have been proposed, such as a hierarchical index of deltas and a memory pool for the overlapping storage of snapshots [13]. Clustering temporally close snapshots and computing a representative for each cluster was also proposed [21], Deltas from representatives are stored for each cluster to achieve high compression. In the G* graph database, snapshots are efficiently stored by taking advantage of commonalities among them [16]. Different versions of each node are stored only once regardless of the number of snapshots it belongs to, and indexed by a compact in-memory index. For load balance and availability snapshot data are replicated among a number of workers.

Historical query processing in these approaches requires as a first and costly step reconstructing the relevant snapshots. Then, queries are processed through an online traversal on each of them. Query performance is addressed by trying to minimize the number of snapshots that need to be reconstructed by minimizing the number of deltas applied [14, 13], avoiding the reconstruction of all snapshots [21], or by parallel query execution and proper snapshot placement and distribution [17]. In this work, we address a different problem, that of indexing for historical reachability queries.

Historical shortest path distance queries were addressed in [9]. The authors propose a method based on ordering nodes or edges pertinent to shortest path computation. Finally, the recent work of [2] also proposes a dynamic indexing scheme for historical distance queries. However, the authors consider only insertions. This assumption simplifies the problem, since two nodes that are reachable remain reachable. The authors propose a dynamic 2hop index construction that is not applicable in the case of node or edge deletions.

Reachability queries on static graphs have been thoroughly investigated along two general directions: transitive closure compression and improving online search.

Transitive Closure Compression. Related research aims at compressing the transitive closure by storing for each node only a subset of the nodes it can reach. The first idea is to decompose the graph in k node-disjoint chains and for each node store only the first node it can reach in each chain [10, 5]. Another line of research extracts a spanning tree of the graph, and uses it to compress the transitive closure. Each node of the tree is labeled with an interval of integers such that if node u is an ancestor of v, the interval of u contains that of v. Reachability through tree edges can be easily determined by a label containment check. To incorporate reachability through non-tree edges each node inherits the intervals of its successors in the graph [1], or a partial transitive closure of non-tree edges is constructed [28]. Building upon the idea of interval labeling, a tree whose vertices are pair-wise disjoint paths extracted from the original graph is used in [12]. Another approach in compressing the transitive closure is 2-hop labeling [7, 22, 6]. Each node stores two sets of intermediate nodes: a set L_{out} of nodes it can reach and a set L_{in} of nodes that can reach it. Node u can reach node v only if $L_{out}(u) \cap L_{in}(v) \neq \emptyset$.

Speeding-up Online Traversal. These methods use interval labeling to aid online traversal by pruning the search space. In [4] and [25], a tree cover of the graph is constructed and then, for the queries that can not be answered by the tree labeling, an online search on the non-tree edges is performed using the labeling to guide the search. In [30], multiple intervals are used for the labeling. If the label containment check does not produce a negative answer, the graph is traversed online using the intervals for pruning the search.

Some of the works discuss the incremental maintenance of the index in the case of evolving graphs [1, 3, 23, 31]. However, the updated index contains reachability information only about the current version of the graph and cannot be used for answering historical queries.

The presented approaches are orthogonal to our approach in that they can be adapted so that they can be used to speed-up or avoid the online traversal of the condensed graph. We have demonstrated this by adapting, one of them, namely 2hop labeling.

3. PROBLEM DEFINITION

Most real world graphs evolve over time as new nodes or edges are added, or existing nodes or edges are deleted. We assume that time is discrete and use successive integers to denote successive time instants. There are two intuitive interpretations of time instants. One interpretation is that of actual time, for example time instant t may correspond to say October 20, 2014, 5:00am PDT. Another view is operational. In this case, time is advanced each time a graph operation, update or delete, occurs. Both interpretations of time instants are consistent with our representation.

Let $G = (V, E)$ be a directed graph where V is the set of nodes and E the set of edges. We use $G_t = (V_t, E_t)$ to denote the *graph snapshot* at time instant t, that is, the set of nodes and edges that exist at time instant t.

DEFINITION 1 (EVOLVING GRAPH). *An evolving graph $\mathcal{G}_{[t_i,t_j]}$ in time interval $[t_i, t_j]$ is a sequence $\{G_{t_i}, G_{t_i+1}, \ldots, G_{t_j}\}$ of graph snapshots.*

An example is shown in Figure 1(a) which depicts an evolving graph $\mathcal{G}_{[t_0,t_3]}$ consisting of four graph snapshots $\{G_{t_0}, G_{t_1}, G_{t_2}, G_{t_3}\}$. For brevity, we denote time instant $t_i + 1$ as t_{i+1} and use t_i and i interchangeably, when the meaning is clear from context.

We use the term *time granularity* to refer to how often a new time instant and the corresponding graph snapshot are created. In the case of actual time, granularity may range for example from milliseconds to years, whereas, in the case of operational time, granularity may be at the level of one or more operations. A fine-grained time granularity necessitates maintaining a large amount of historical information, but supports precise historical queries.

Given a static directed graph $G = (V, E)$ and two nodes u, $v \in V$, a *reachability query* asks whether there exists a path from u to v in G. For evolving graphs, we introduce the following two types of historical reachability queries.

DEFINITION 2 (HISTORICAL REACHABILITY QUERY). *Let $\mathcal{G}_{[t_i,t_j]} = \{G_{t_i}, G_{t_i+1}, \ldots G_{t_j}\}$, be an evolving graph, $I_Q = [t_k, t_l] \subseteq [t_i, t_j]$ a time interval and v, u a pair of nodes:*

(i) a conjunctive historical reachability query $u \overset{I_{Q_\wedge}}{\rightsquigarrow} v$ returns true, if there exists a path from u to v in all graph snapshots G_{t_m}, $t_k \leq t_m \leq t_l$ of $\mathcal{G}_{[t_i,t_j]}$.

(ii) a disjunctive historical reachability query $u \overset{I_{Q_\vee}}{\rightsquigarrow} v$ returns true, if there exists a path from u to v in at least one graph snapshot G_{t_m}, $t_k \leq t_m \leq t_l$, of $\mathcal{G}_{[t_i,t_j]}$.

Our goal is to derive methods for answering reachability queries efficiently. A straightforward solution would be to build a different index for each of the graph snapshots and then pose a reachability query at each one of them. However, this solution imposes large space overheads. In addition, it requires extra processing for combining the results of each query. Instead, we propose building indexes for intervals.

4. VERSION GRAPH

In this section, we present the version graph, a natural concrete representation of an evolving graph. First, let us define the notion of lifespan. For a node u (or, edge e), its lifespan denotes the set of time intervals during which u (resp. e) existed in an evolving graph. More formally, given an evolving graph $\mathcal{G}_I = \{G_{t_i}, G_{t_i+1}, \ldots, G_{t_j}\}$, the *lifespan*, $\mathcal{L}(u)$, (resp. $\mathcal{L}(e)$) of a node u (resp. edge e) is a set of intervals such that an interval $[t_i, t_j] \subseteq I$ belongs to $\mathcal{L}(u)$, (resp. $\mathcal{L}(e)$), if and only if, for all $t_i \leq t_m \leq t_j$, $u \in V_{t_m}$ (resp. $e \in E_{t_m}$).

We model lifespans as sets of time intervals to capture the general case of graph evolution, where nodes and edges may be deleted and then re-inserted at subsequent snapshots. Set of time intervals are also known as *temporal elements* [11]. If we do not allow deleted nodes or edges to be re-inserted, then lifespans are just intervals. Furthermore, if there are no deletions, all lifespans are intervals of the form $[t_i, t_{curr}]$, where t_i is the time instant the node or edge first appeared and t_{curr} is the time instant of the current snapshot. Therefore, in this case, lifespans can be represented simply by the time instant t_i. In the following, we use I to denote time intervals and $\mathcal{I}$ to denote sets of time intervals. To represent an evolving graph $\mathcal{G}_I$, we use a *version graph* VG_I. A version graph is a labeled directed graph that captures the evolution of the graph in a concise manner.

DEFINITION 3 (VERSION GRAPH). *Given an evolving graph $\mathcal{G}_I = \{G_{t_i}, G_{t_i+1}, \ldots, G_{t_j}\}$, its version graph is an edge and node labeled, directed graph $VG_I = (V_I, E_I, \mathcal{L}_u, \mathcal{L}_e)$ where: $V_I = \bigcup_{t_m \in I} V_{t_m}$, $E_I = \bigcup_{t_m \in I} E_{t_m}$, $\mathcal{L}_u : V_I \rightarrow \mathcal{I}$ assigns to each node u in V_I its lifespan $\mathcal{L}_u(u)$ and $\mathcal{L}_e : E_I \rightarrow \mathcal{I}$ assigns to each edge e in E_I its lifespan $\mathcal{L}_e(e)$.*

An example is shown in Figure 1(b) which depicts the version graph for the evolving graph in Figure 1(a).

4.1 Lifespan Operations

Let us define a number of operations on lifespans, i.e., set of intervals. For two sets $\mathcal{I}$ and $\mathcal{I}'$ of time intervals, we say that $\mathcal{I}$ *covers* $\mathcal{I}'$, denoted $\mathcal{I} \sqsupseteq \mathcal{I}'$, if for each time instant t in an interval I' of $\mathcal{I}'$, there is an interval I in $\mathcal{I}$ such that t belongs to I. We also use $\mathcal{I} \sqsupseteq I$ for an interval I and $\mathcal{I} \sqsupseteq t$ for a time instant t. We say that two sets $\mathcal{I}$ and $\mathcal{I}'$ of time intervals are equivalent, $\mathcal{I} \approx \mathcal{I}'$, if $\mathcal{I} \sqsupseteq \mathcal{I}'$ and $\mathcal{I}' \sqsupseteq \mathcal{I}$.

We would like to maintain the smallest among equivalent sets of intervals. We call such sets *minimum* sets. Let us first define some simple properties for time intervals. Two time intervals $I = [t_i, t_j]$ and $I' = [t'_i, t'_j]$ are called *disjoint*, when $I \cap I' = \emptyset$ and *overlapping* otherwise. They are called *continuous* when $t'_i = t_j + 1$ and non-continuous otherwise. It is easy to see that the following proposition holds.

PROPOSITION 1.

(i) A set of intervals is minimum, if and only if, it consists of disjoint and non-continuous intervals.

(ii) For each set of time intervals, there is a unique equivalent minimum interval set.

We next define two useful operations on interval sets, namely, *join* and *merge*. Given two sets of intervals, join returns the time instants common to both, while merge returns the time instants present in at least one of them.

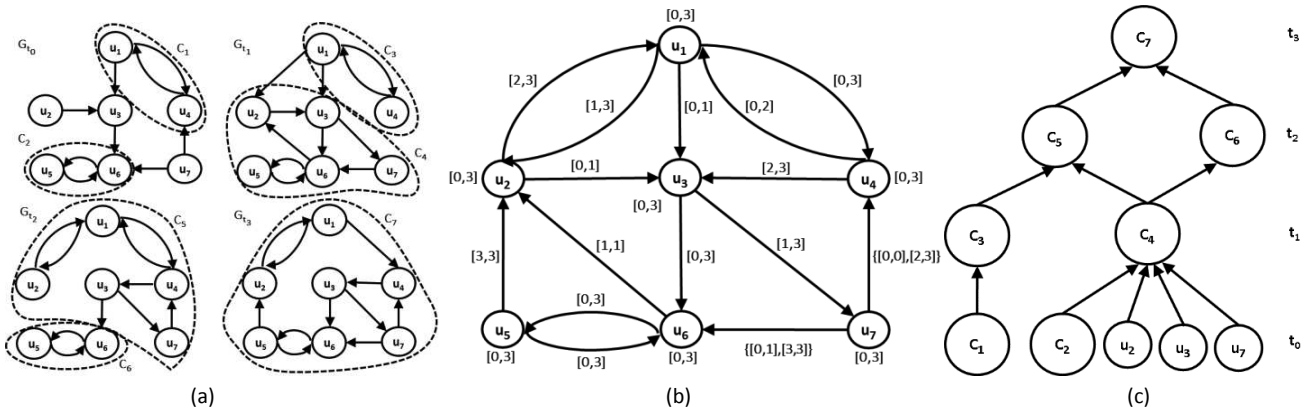


Figure 1: Example of (a) an evolving graph, (b) the corresponding version graph, (c) SCC evolution

Definition 4 (Join and Merge of Interval Sets). *Let $\mathcal{I} = \{I_1, \ldots I_k\}$ and $\mathcal{I}' = \{I'_1, \ldots I'_l\}$ be two sets of time intervals.*

(i) Join $\mathcal{I} \otimes \mathcal{I}'$ of $\mathcal{I}$ and $\mathcal{I}'$ is the minimum set equivalent to $\{I_1 \cap I'_1, \ldots I_1 \cap I'_l, \ldots, I_k \cap I'_1, \ldots I_k \cap I'_l\}$.

(ii) Merge $\mathcal{I} \oplus \mathcal{I}'$ of $\mathcal{I}$ and $\mathcal{I}'$ is the minimum set equivalent to $\mathcal{I} \cup \mathcal{I}'$.

Note that if $\mathcal{I}$ and $\mathcal{I}'$ are minimum, then the set $\{I_1 \cap I'_1, \ldots I_1 \cap I'_l \ldots, I_k \cap I'_1 \}$ is a minimum set, whereas the set $\{I_1 \cup I'_1, \ldots I_1 \cup I'_l, \ldots, I_k \cup I'_1 \ldots I_k \cup I'_l\}$ may not be minimum.

The lifespan $\mathcal{L}(p)$ of a path p includes the time intervals during which all its edges coexist. Clearly, for a path $p = e_1 \ldots e_m$, it holds that $\mathcal{L}(p) = \mathcal{L}_e(e_1) \otimes \ldots \otimes \mathcal{L}_e(e_m)$, where $\mathcal{L}_e(e_i)$, $1 \leq i \leq m$, is the lifespan of e_i. For example, for path $p = ((u_4, u_3), (u_3, u_7), (u_7, u_6))$ in Figure 1(b), $\mathcal{L}(p)$ = {[2,3]} $\otimes$ {[1,3]} $\otimes$ {[0,1], [3, 3]} = {[3,3]}, while for path p'= $((u_1, u_3), (u_3,u_7), (u_7,u_4))$, $\mathcal{L}(p')$ = {[0,1]} $\otimes$ {[1,3]} $\otimes$ {[0,0], [2,3]} = $\emptyset$.

We can now define the lifespan, $\mathcal{L}(u, v)$, of the reachability between two nodes u and v. Let $P(u, v) = \{p_1, \ldots p_l\}$ be the set of all paths from u to v. $\mathcal{L}(u, v)$ depends on the lifespans of all possible paths in VG_I from u to v, in particular, $\mathcal{L}(u, v) = \mathcal{L}(p_1) \oplus \ldots \oplus \mathcal{L}(p_l)$. For example, for nodes u_4 and u_6 in Figure 1(b), $P(u_4, u_6) = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ where $p_1 = u_4u_3u_6$, $p_2 = u_4u_3u_7u_6$, $p_3 = u_4u_1u_3u_6$, $p_4 = u_4u_1u_3u_7u_6$, $p_5 = u_4u_1u_2u_3u_6$, $p_6 = u_4u_1u_2u_3u_7u_6$ (note, that for notational brevity, paths were denoted by the participating nodes instead of edges). Then, $\mathcal{L}(u_4, u_6) = \{[2, 3]\}$ $\oplus$ $\{[3, 3]\}$ $\oplus$ $\{[0, 1]\}$ $\oplus$ $\{[1, 1]\}$ $\oplus$ $\{[1, 1]\}$ $\oplus$ $\{[1, 1]\}$ = $\{[0, 3]\}$.

Clearly, historical reachability queries can be represented in terms of lifespans. Specifically, given a version graph VG_I, a time interval $I_Q = [t_k, t_l] \subseteq [t_i, t_j]$ and two nodes v, u,

(i) a conjunctive historical reachability query $u \overset{I_{Q\wedge}}{\rightsquigarrow} v$ returns true, if and only if, $\{I_Q\} \otimes \mathcal{L}(u, v) \sqsupseteq I_Q$.

(ii) a disjunctive historical reachability query $u \overset{I_{Q\vee}}{\rightsquigarrow} v$ returns true, if and only if, $\{I_Q\} \otimes \mathcal{L}(u, v) \neq \emptyset$.

To represent lifespans, we use bit arrays. Assume without loss of generality, that the maximum time instant, that is, the number of graph snapshots, is T. Then, a lifespan, i.e., set of intervals, $\mathcal{I}$ is represented by a bit array B of size T, such that $B[i] = 1$ if $\mathcal{I} \sqsupseteq i$, and 0, otherwise. For example, take $\mathcal{I} = \{[2, 4], [9, 10], [13, curr]\}$ and $T = 16$. The bit array representation of $\mathcal{I}$ is 00111000011001111. This leads to an efficient implementation of both join $\otimes$ and merge $\oplus$. In particular, let $\mathcal{I}$ and $\mathcal{I}'$ be two set of intervals and B and B' be their bit arrays. Then, $\mathcal{I} \otimes \mathcal{I}'$ is computed as B logical-AND B' and $\mathcal{I} \oplus \mathcal{I}'$ as B logical-OR B'. An alternative representation would be to use ordered lists of intervals. Lifespan operations would then be performed using variations of merge sort resulting in $O(T)$ complexity. Lists impose in general large computational overheads in computing reachability.

4.2 Baseline Approaches

There are two baseline approaches to answering reachability queries on static graphs, namely pre-computation of the graph transitive closure and online traversal of the graph. In this section, we revisit these baseline approaches for historical reachability queries on a version graph.

4.2.1 Historical Transitive Closure

Instead of maintaining a different transitive closure for each graph snapshot of the evolving graph $\mathcal{G}_I$, we maintain a single transitive closure, CL_I for the version graph VG_I. The transitive closure includes for each pair of nodes u, v, their reachability lifespan, $\mathcal{L}(u, v)$. To construct the transitive closure, we use a variation of the Floyd-Warshall algorithm that takes into account lifespans, shown in Algorithm 1. If there is a path $p_{u,w}$ from node u to node w and a path $p_{w,v}$ from node w to node v then there exists a path $p_{u,v} = (p_{u,w}, p_{w,v})$ from u to v with $\mathcal{L}(p_{u,v}) = \mathcal{L}(p_{u,w}) \otimes \mathcal{L}(p_{w,v})$ and $\mathcal{L}(p_{u,v})$ is merged with the $\mathcal{L}(u, v)$ computed so far.

The time complexity for Algorithm 1 is $O(|V_I|^3 T)$ in the worst case and requires storage in the order of $|V_I|^2$. For answering a reachability query $u \overset{I_{Q\vee}}{\rightsquigarrow} v$ or $u \overset{I_{Q\wedge}}{\rightsquigarrow} v$, initially the entry $\mathcal{L}(u, v)$ in CL_I is located and then joined with the query interval I_Q, thus requiring constant time complexity.

4.2.2 Online Traversal of the Version Graph

A straightforward approach to process a reachability query for an interval I_Q would be to perform an online traversal on all graph snapshots G_t, $t \in I_Q$. When using the version graph representation, this corresponds to traversing

Algorithm 1 TransitiveClosure(VG_I)

Input: Version graph VG_I
Output: The transitive closure CL_I

```
1: for all u, v ∈ V_I × V_I do
2:    if (u, v) ∈ E_I then
3:       CL_I(u, v) = L_e((u, v))
4:    else
5:       CL_I(u, v) = ∅
6:    end if
7: end for
8: for w = 1 to |V_I| do
9:    for all u, v ∈ V_I × V_I do
10:      CL_I(u, v) = CL_I(u, v) ⊕ (CL_I(u, w) ⊗ CL_I(w, v))
11:   end for
12: end for
```

only edges e such that $\mathcal{L}_e(e) \sqsupseteq t$, once for each $t \in I_Q$. We call this approach, *instant based traversal*.

To avoid multiple traversals, i.e., one for each snapshot in I_Q, we consider an *interval based traversal* of the version graph. The BFS-based interval traversal for disjunctive historical queries is shown in Algorithm 2 and for conjunctive historical queries in Algorithm 3.

In particular, for conjunctive queries, since a node v may be reachable from u through different paths at different graph snapshots, we maintain an interval set $\mathcal{R}$ with the part of $\mathcal{L}(u, v) \otimes I_Q$ covered so far (line 9, Algorithm 3). The traversal ends when $\mathcal{R}$ covers the whole query time interval I_Q (line 10, Algorithm 3).

To speed-up traversal, we perform a number of pruning tests. The traversal stops when we reach a node whose lifespan is outside the query interval. In addition, the traversal stops at a neighbor w of a node n when $\{I_Q\} \otimes \mathcal{L}_e(n, w) \neq \emptyset$ since a node v cannot be reachable through an edge which is not alive in at least one t inside the query interval (line 6, Algorithms 2 and 3).

Still an edge may be traversed multiple times, if it participates in multiple paths from source to target. To reduce the number of such traversals, we provide additional pruning by recording for each node w, an interval set $\mathcal{IN}(w)$ with the parts of the query interval for which it has already been traversed. If the query reaches w again looking for interval I' $\subseteq I_Q$ and $\mathcal{IN}(w) \sqsupseteq I'$, the traversal is pruned (line 11 of Algorithm 2, line 15 of Algorithm 3).

For example, consider the version graph in Figure 1(b) and query $u_1 \overset{[0,3]\wedge}{\rightsquigarrow} u_5$. Paths $p_1 = u_1u_3u_6u_5$, $p_2 = u_1u_3u_7u_6u_5$, $p_3 = u_1u_2u_3u_6u_5$, $p_4 = u_1u_2u_3u_7u_6u_5$, $p_5 = u_1u_4u_3u_6u_5$ and $p_6 = u_1u_4u_3u_7u_6u_5$ with $\mathcal{L}(p_1) = \{[0, 1]\}$, $\mathcal{L}(p_2) = \{[1, 1]\}$, $\mathcal{L}(p_3) = \{[1, 1]\}$, $\mathcal{L}(p_4) = \{[1, 1]\}$, $\mathcal{L}(p_5) = \{[2, 3]\}$ and $\mathcal{L}(p_6)$ $= \{[3, 3]\}$ need to be traversed to conclude correctly that the result of the query is true. Hence, some edges, e.g., (u_3, u_6), (u_6, u_5) need to be traversed multiple times for different time intervals $I'_i \subseteq I_Q$. However, when the query reaches u_3 again through path p_3, it is pruned and it does not traverse the edge (u_3, u_6) since $\mathcal{IN}(u_3)$ is equal to {[0,1]} which covers the current query interval $I' = \{[1,1]\}$.

Since in the worst case for both instant and interval based traversal each edge may be traversed $|I_Q|$ times, the complexity for both traversals is $O((|V_I| + |E_I|)|I_Q|)$. However, in practice interval based traversal outperforms the instant based one since each edge traversal covers large parts of the query interval instead of a single time instant. Furthermore, pruning guarantees that an edge will not be traversed twice for the same interval.

Algorithm 2 Disjunctive-BFS(VG_I, u, v, $\{I_Q\}$)

Input: Version graph VG_I, nodes u, v, interval $I_Q \subseteq I$
Output: True if v is reachable from u in any time instant in I_Q and false otherwise

```
1: create a queue N, create a queue INT
2: enqueue u onto N, enqueue I_Q onto INT
3: while N ≠ ∅ do
4:    n ← N.dequeue()
5:    i ← INT.dequeue()
6:    for all w s.t. (n, w) in VG_I and {I_Q} ⊗ L_e((n, w))
      ≠ ∅ do
7:       if w == v then
8:          Return(true)
9:       end if
10:      I' = {I_Q} ⊗ L_e(u, w)
11:      if IN(w) ⋢ I' then
12:         IN(w) = IN(w) ⊕ I'
13:         enqueue w onto N
14:         enqueue I' onto INT
15:      end if
16:   end for
17: end while
18: Return(false)
```

5. THE TIMEREACH INDEX

Our approach exploits the fact that many real-world social graphs are characterized by large strongly connected components (SCC) [20, 15]. Thus, instead of maintaining reachability information for pairs of nodes, we maintain information about the SCCs that each node belongs to. If two nodes belong to the same component, then they are reachable. However, as the graph evolves over time, its strongly connected components change as well. An example is shown in Figure 1(c) that depicts the SCCs of the graph in Figure 1(b) as they evolve over time.

Given an evolving graph $\mathcal{G}_I = \{G_{t_i}, G_{t_i+1}, \ldots, G_{t_j}\}$, we invoke at each graph snapshot $G_{t_k} \in \mathcal{G}_I$ an algorithm, e.g., Tarjan's algorithm [24], to identify the corresponding set of SCCs. A unique id is assigned to each SCC at each snapshot.

For each node u, we maintain a list $P(u)$ that contains (C, t) pairs specifying the strongly connected component C to which node u belongs at time instant t. $P(u)$ is called *posting list* and each pair in the list a *posting*. The storage complexity is $\Omega(|V_I||I|)$, since each node participates in at most one SCC at each time instant. If we use Tarjan's algorithm [24], the time complexity for constructing the lists is $O((|V_I| + |E_I|)|I|)$, since each run of the Tarjan's algorithm has an $O(|V_I| + |E_I|)$ complexity.

For presentation clarity, we assume that single nodes form singleton SCCs whose ids are the ids of the corresponding nodes. However, for space efficiency, we do not maintain postings in this case.

We perform an additional optimization. Many nodes have strong connections, i.e. they remain in the same components even in the face of component splits and joins. We exploit this fact to reduce the storage space required for the postings by observing that the posting lists of these nodes consist of

Algorithm 3 Conjunctive-BFS(VG_I, u, v, $\{I_Q\}$)

Input: Version graph VG_I, nodes u, v, interval $I_Q \subseteq I$
Output: True if v is reachable from u in all time instants in I_Q and false otherwise

```
1: create a queue N, create a queue INT
2: enqueue u onto N, enqueue I_Q onto INT
3: while N ≠ ∅ do
4:    n ← N.dequeue()
5:    i ← INT.dequeue()
6:    for all w s.t. (n, w) in VG_I and {I_Q} ⊗ L_e((n,w)) ≠ ∅ do
7:       I' = {I_Q} ⊗ L_e(n,w)
8:       if w == v then
9:          R = R ⊕ I'
10:         if R ⊒ I_Q then
11:            Return(true)
12:         end if
13:         continue
14:      end if
15:      if IN(w) ⋢ I' then
16:         IN(w) = IN(w) ⊕ I'
17:         enqueue w onto N
18:         enqueue I' onto INT
19:      end if
20:   end for
21: end while
22: Return(false)
```

the same elements. We avoid redundancy by storing such lists only once and replacing the posting lists of the relevant nodes with pointers to the common list. We call this approach *posting sharing*.

An example is shown in Figure 2(a), where, for instance, the first posting list indicates that nodes with ids 1 up to 50 belong to the strongly connected component with id C_1 at time t_0, C_6 at t_1 and C_9 at t_2.

In addition, for each graph snapshot G_{t_k}, we construct a SCC graph snapshot $G_{S_{t_k}} = (V_{S_{t_k}}, E_{S_{t_k}})$ such that there is a node U in $V_{S_{t_k}}$ for each SCC in G_{t_k}, and there is an edge (U, V) in $E_{S_{t_k}}$, if and only if, there is an edge (u, v) in G_{t_k} from a node u that belongs to the SCC that corresponds to U to a node v that belongs to the SCC that corresponds to V. For a time interval $I = [t_i, t_j]$, this results in an evolving SCC graph $\mathcal{G}_{S_I} = \{G_{S_{t_i}}, G_{S_{t_i+1}}, \ldots, G_{S_{t_j}}\}$. We construct the SCC graphs incrementally, as the SCCs are created. The size of each SCC graph depends on the size of the original snapshot graph and in the worst case is equal to it.

We call this approach simple *TimeReach* (TR). To answer a reachability query $u \overset{I_{Q_\wedge}}{\rightsquigarrow} v$, (or, $u \overset{I_{Q_\vee}}{\rightsquigarrow} v$), we check for each $t \in I_Q$ whether u and v belong to the same component. If this is not the case, we traverse the corresponding G_{S_t}.

Next, we present a more space efficient method of exploiting strongly connected components for historical queries.

5.1 Condensed TimeReach

While in the TR approach, we maintain information per time instant, we would like to aggregate such information to express SCC participations during time intervals. In this case, a posting (C, I'), $I' \subseteq I$, belongs to $P(u)$, if u participates in the SCC with id C at all time instants in I'. Our goal is to minimize the total number of such postings.

Problem 1 (Optimal SCC-id assignment). *Given a time interval I and a set of SCCs for each $t \in I$, find an assignment of ids to SCCs that results in the minimum number of postings.*

A new posting is created, each time a node participates in a different SCC. Thus, SCC ids should be re-assigned so that the number of such new postings is minimized. We use a weighted graph to formalize the optimal assignment of ids to SCCs.

In particular, we model SCC evolution over a time interval I using a weighted graph $G_C(V_C, E_C, \mathcal{W})$ where each node $U \in V_C$ corresponds to a SCC that existed at some time instant $t \in I$, and an edge $e = (U, V) \in E_C$, if and only if, SCC U existed at time t_k, SCC V existed at time $t_k + 1$ and there is at least one node that belongs to both U and V. $\mathcal{W}$ assigns to each edge $e = (U, V)$ a weight $\mathcal{W}(e)$ that corresponds to the number of nodes that belong to both U and V.

An example of a weighted graph is shown in Figure 2(b) that depicts the evolution of the graph whose posting lists are shown in Figure 2(a). For instance, component C_7 created at time instant t_1 consists of 100 nodes from component C_4 and 150 nodes from C_5.

Let $G_{C_{[t_k,t_k+1]}}(V_{C_{[t_k,t_k+1]}}, E_{C_{[t_k,t_k+1]}}, \mathcal{W})$ be the subgraph of $G_C(V_C, E_C, w)$, that consists of the nodes $U \in V_{C_{[t_k,t_k+1]}}$ that correspond to the SCCs that exist at time interval $[t_k, t_k + 1]$. $G_{C_{[t_k,t_k+1]}}$ represents one step in the SCC evolution. Note that, from the definition of G_C, $G_{C_{[t_k,t_k+1]}}$ is a bipartite graph.

We make the following observation. At time instant $t_k + 1$, a new posting is created exactly for those nodes that participated in a different SCC at $t_k + 1$ than at t_k. The number of these new postings is equal to the sum of weights from node U to V in $G_{C_{[t_k,t_k+1]}}$ where U has a different id than V. Thus, to minimize the number of new postings, we have to maximize the weight of the edges between pairs of nodes that have the same id. This corresponds to finding a maximum bipartite matching of $G_{C_{[t_k,t_k+1]}}$.

Theorem 1. *The optimal SCC-id assignment problem can be reduced to the problem of finding the maximum weight bipartite matching (MWM) M_k of each $G_{C_{[t_k,t_k+1]}}$.*

Proof. As shown above, solving the MWM for each bipartite graph $G_{C_{[t_k,t_k+1]}}$ minimizes the number of new postings created at $t_k + 1$. We shall show that this step-wise assignment is optimal overall in G_C. For the purposes of contradiction, assume that the optimal assignment is a set N of edges, $N \subset E_C$ and that N is different from the set of edges attained through the maximum bipartite matchings, that is, $\sum_{e \in N} w(e) > \sum_k \sum_{e \in M_k} w(e)$. Hence, for some m, for $N_m = N \cap E_{C_{[t_m,t_m+1]}}$ it holds that $\sum_{e \in N_m} w(e) > \sum_{e \in M_m} w(e)$, which means that M_m is not a MWM, which is a contradiction. □

Figure 2(c) shows the weighted graph after the assignment of new ids through bipartite matching, while Figure 2(d) shows the new posting lists.

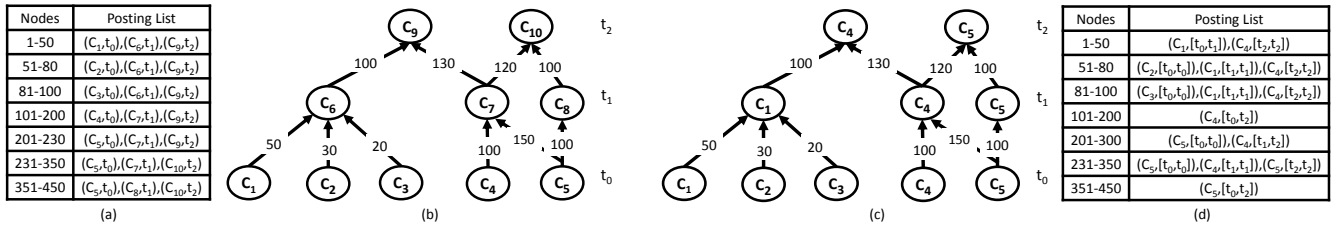


Figure 2: (a) Shared posting lists, (b) weighted graph modeling the evolution of SCCs, (c) weighted graph after the bipartite matching, and (d) the compressed shared posting lists

The maximum weight bipartite matching problem is well-studied (e.g., see [8] for a survey). The most widely used algorithm for solving this problem on a graph $G(V, E)$ is the Hungarian algorithm whose running time ranges from $O(|V|^3)$ to $O(|E||V| + |V|^2 loglog|V|)$ depending on the implementation. Another category of algorithms depends on the edge weights and the fastest one runs in $O(|E|\sqrt{|V|}logW)$ time, where W is the maximum edge weight. In addition, a number of fast approximation algorithms have been proposed. The simplest such algorithm is the greedy algorithm that sorts the edges by weight and repeatedly picks the edge with the largest weight. This algorithm can be implemented with $O(|E|)$ time complexity and produces a 1/2 worst case approximation.

The incremental algorithm for constructing the SCC postings is presented in Algorithm 4. It takes as input the current snapshot and the postings computed up to the previous snapshot, and constructs the current postings. It starts by computing the SCCs using Tarjan's algorithm with complexity $O(|V_t| + |E_t|)$ (line 2). Then, it constructs the graph $G_{C_{[t,t+1]}}$ with complexity $O(|E_{C_{[t-1,t]}}|)$ (line 5). Next, the MWM is computed and new ids are assigned to the new SCCs (lines 6 - 9). The complexity of this step depends on which algorithm is used for computing the MWM. We use the greedy algorithm with complexity $O(|E_{S_{[t-1,t]}}|)$. Finally, the SCC postings are created/updated for each node of the current snapshot, creating a new entry only for nodes that participate in a different SCC (with a different id) than the one in time instant $t-1$ (lines 11 - 22). The complexity of these steps is $O(|V_t|)$ since each operation in the loop has constant time complexity. Thus, in total the running time of the algorithm is $O(|V_t| + |E_t|)$.

As in the simple TR approach, we also construct the evolving SCC graph, which in this case has a much smaller number of nodes due to the reduction of the number of strongly connected components achieved by the bipartite matching.

Finally, we construct the version graph $VG_{S_I} = (V_{S_I}, E_{S_I}, \mathcal{L}_u, \mathcal{L}_e)$ of the evolving SCC graph that we call *condensed version graph*. We construct the condensed version graph incrementally as follows. For each snapshot $G_{t_i} \in \mathcal{G}_I$, for each edge $(u, v) \in E_{t_i}$ we look up the postings $P(u)$, $P(v)$ for entries (U, I'), (V, I'') s.t. $t_i \in I'$ and $t_i \in I''$. If $U \neq V$ and edge $(U, V) \notin E_{S_I}$, the edge is added with lifespan $\{[t_i, t_i]\}$, otherwise the lifespan of the edge is extended to include t_i. We call the above approach *condensed TimeReach* (TRC).

5.2 Query Processing

Query processing of a (disjunctive or conjunctive) reachability query $u \overset{I_Q}{\rightsquigarrow} v$ is performed in two steps. In the first step, the appropriate postings of nodes u and v are retrieved. If the two nodes belong to the same strongly connected component during the whole query interval for conjunctive queries or once for disjunctive queries, the answer is true. Otherwise, let $\mathcal{I}'_\mathcal{Q}$ be the set of intervals during which nodes u and v belong to different components. The query is re-written as a set of reachability sub-queries of the form $U_k \overset{I_{Q_i}}{\rightsquigarrow} V_m$, where u belongs to SCC U_k and v belongs to SCC V_m for some common time interval I_{Q_i}, $\mathcal{I}'_\mathcal{Q} \sqsupseteq I_{Q_i}$, the set $\mathcal{I}_\mathcal{Q} = \bigcup_i I_{Q_i}$ consists of disjoint intervals, and $\mathcal{I}_\mathcal{Q} \approx \mathcal{I}'_\mathcal{Q}$. The results of the sub-queries are combined to produce the answer for the query through an AND (OR) for conjunctive (disjunctive) queries.

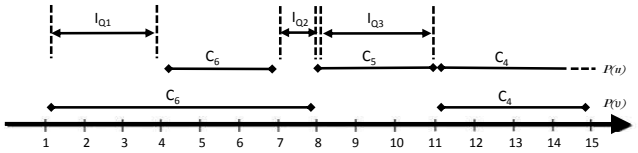


Figure 3: Example of splitting query $u \overset{[1,15]\wedge}{\rightsquigarrow} v$

For example, consider the query $u \overset{[1,15]\wedge}{\rightsquigarrow} v$ in Figure 3, where the posting lists for u and v are respectively, $P(u) = (C_6\ [4, 7],\ C_5\ [8, 11],\ C_4\ [11, curr]$ and $P(v) = (C_6\ [1, 8],\ C_4\ [11, 15])$. The query is split in three sub-queries: $u \overset{I_{Q_1}\wedge}{\rightsquigarrow} C_6$, $u \overset{I_{Q_2}\wedge}{\rightsquigarrow} C_6$, $v \overset{I_{Q_3}\wedge}{\rightsquigarrow} C_5$.

In the worst case, the two nodes belong to a different SCCs at each time instant in I_Q, thus we need to traverse the condensed version graph for each t with a cost of $O(|I_Q|(|V_{S_I}| + |E_{S_I}|))$ Two factors that influence performance are the number of postings for each node and the size of the condensed version graph. The smaller the number of postings, the fewer sub-queries are required in the second step. The smaller the condensed version graph, the faster the traversals. Hence, the optimal assignment of SCC ids is crucial to query processing performance, since it keeps the posting lists short and the size of the condensed version graph small.

5.3 Interval 2Hop

Reachability on version graphs can be made more efficient by maintaining additional information. In this paper, we use an approach based on pruned landmark 2hop labeling [2, 29]. The idea is that for each node u of a given graph, we maintain two labels $L_{in}(u)$ and $L_{out}(u)$ which include nodes that can reach u and can be reached by u respectively. The labels are computed such that a node u reaches v, if an only if, $L_{in}(v) \cap L_{out}(u) \neq \emptyset$. Instead of traversing the graph, a reachability query can now be answered by using the labels.

For historical reachability queries, we also keep along with each node w in $L_{in}(v)$ the reachability lifespan $\mathcal{L}(w, v)$ and along with each node w in $L_{out}(u)$ the reachability lifespan

Algorithm 4 ConstructSccPostings(G_t, P_{t-1}, $G_{S_{[t-2,t-1]}}$)

Input: Snapshot G_t, SCC postings P_{t-1}
Output: SCC postings P_t

1: $S_{SCC_t} = \emptyset$, $M = \emptyset$
2: Run Tarjan's algorithm on G_t
3: S_{SCC_t} is the set of the detected SCCs where each $SCC_i \in S_{SCC_t}$ is assigned a unique id C_i
4: **if** $t > 0$ **then**
5: Construct $G_{S_{[t-1,t]}}$ from S_{SCC_t} and $G_{S_{[t-2,t-1]}}$
6: Compute maximum weight matching M
7: **for all** edges $e = (U, V) \in M$ **do**
8: $C_v = C_u$
9: **end for**
10: **end if**
11: **for all** nodes $u \in V_t$ **do**
12: find $SCC_i \in S_{SCC_t}$ s.t. $u \in SCC_i$
13: **if** $P_{t-1}(u) \neq \emptyset$ **then**
14: **if** $P_{t-1}(u)[end].C \neq C_i$ **then**
15: $P_{t-1}(u)[end].I = [t_s, t-1]$
16: $P_{t-1}(u).add(C_i, [t, curr])$
17: **end if**
18: **else**
19: $P_{t-1}(u).add(C_i, [t, curr])$
20: **end if**
21: **end for**
22: $P_t = P_{t-1}$

$\mathcal{L}(u, w)$. In the presence of 2hop labels, to answer a query $u \overset{I_Q^\wedge}{\rightsquigarrow} v$ ($u \overset{I_Q^\vee}{\rightsquigarrow} v$), we compute the set $L_{in}(v) \cap L_{out}(u)$ and then for each w in $L_{in}(v) \cap L_{out}$, we join the lifespan of w in $L_{in}(v)$ with the lifespan of w in $L_{out}(u)$. To answer the query the joined lifespans $\mathcal{L}(w)$ of nodes w in $L_{in}(v) \cap L_{out}$ are joined with the query interval $\mathcal{L}$ to see whether they cover I_Q (or, have at least a time instant in common).

We compute the labels for the nodes of the condensed version graph, incrementally. For an interval $I = [t_i, t_j]$, we compute the labels for the SCC graph snapshots at each time t in I, starting from t_i. For each time t_k, $t_k > t_i$, we merge the labels computed for a node C at time t_k, with the labels computed for C at the previous time $t_k - 1$. For the construction of L_{in} and L_{out} for each SCC graph snapshot at time instant t_k, we process the nodes of the graph by using the $INOUT$ strategy that starts a BFS traversal from the nodes with the largest $(indegree(u)+1) \times (outdegree(u)+1)$ [29]. An example of the final 2hop labels of each SCC node in a version graph is given in Figure 4.

6. EXPERIMENTAL EVALUATION

To evaluate our approach, we used three real datasets: Facebook (FB) [27], Flickr (FL) [19] and YouTube (YT) [18]. The characteristics of each dataset are shown in Table 3. For example, FB consists of 871 daily snapshots of the New Orleans Facebook friendship graph, which correspond to 125 weekly or 29 monthly snapshots. We report the number of nodes, edges, and SCCs (singleton SCCs are not included) and the size of the largest SCC at the first and last snapshot.

All three datasets are treated as directed. Also, all datasets are insert-only, i.e. they do not contain information about node/edge deletions. Therefore, we synthetically generate random edge deletes. The input parameters and their default values are shown in Table 1.

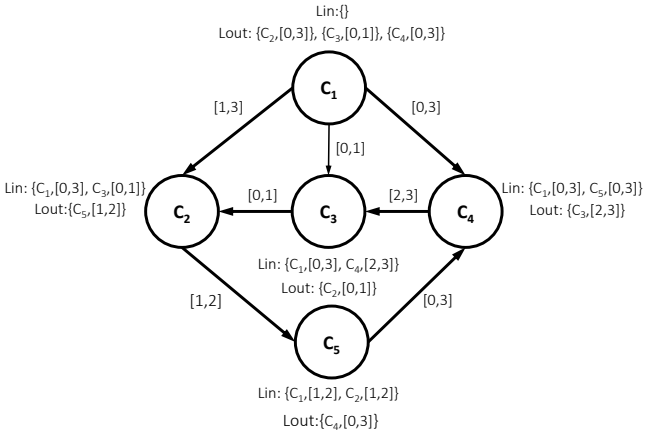


Figure 4: An example of interval 2hop labels

We evaluate the size and the construction time of the Version Graph (VG), the Transitive Closure (TC), the simple TimeReach (TR), the condensed TimeReach (TRC) and the condensed TimeReach with 2hop labels (TRCH). We also evaluate the online processing of historical reachability queries using an instant-based (INS) or interval-based (INT) traversal of the version graph and using the various TimeReach indexes. Table 2 summarizes the various approaches.

We ran our experiments on a system with a quad-core Intel Core i7-3820 3.6 Ghz processor and 64 GB memory. We only used one core for all experiments.

Table 1: Input parameters

		# of nodes	Snapshot granularity	Query interval (in days)	% of deletes
FB	Default	61,096	day	7	10
	Range	117 - 61,096	day, week, month	7 - 35	0 - 30
YT	Default	1,138,499	day	7	10
	Range	1,004,777 - 1,138,499	day, week, month	7 - 35	0 - 30
FL	Default	2,302,925	day	7	10
	Range	1,487,058 - 2,302,925	day, week, month	7 - 35	0 - 30

6.1 Index Size

In the first set of experiments, we evaluate the various approaches in terms of their storage requirements. The size of the TR and TRC include the storage required for maintaining the posting lists and the SCC graphs, while the size of the TRCH includes in addition the storage required for the 2hop labels.

Table 2: Overview of difference approaches

VG	Version Graph
TC	Transitive Closure
TR	(Simple) TimeReach
TRC	Condensed TimeReach
TRCH	Condensed TimeReach with 2hop labels
INS	Instant-based traversal of the version graph
INT	Interval-based traversal of the version graph

Table 3: Dataset properties

	Snapshot Granularity	# nodes		# edges		# SCC		Max SCC (# nodes)	
		first	last	first	last	first	last	first	last
FB	(daily) 871	117	61,096	128	1,139,081	10	374	3	51,286
	(weekly) 125	1,429	61,096	2,365	1,139,081	138	374	18	51,286
	(monthly) 29	4,239	61,096	12,224	1,139,081	279	374	96	51,286
YT	(daily) 37	1,004,777	1,138,499	4,379,283	4,452,646	9,807	11,360	457,932	509,332
	(weekly) 6	1,025,536	1,138,499	4,379,283	4,452,646	9,807	11,360	465,668	509,332
	(monthly) 2	1,116,602	1,138,499	4,446,042	4,452,646	10,664	11,360	485,273	509,332
FL	(daily) 134	1,487,058	2,302,925	17,022,083	33,140,018	42,163	58,636	1,004,426	1,605,184
	(weekly) 20	1,507,700	2,302,925	17,393,321	33,140,018	42,163	58,636	1,010,498	1,605,184
	(monthly) 5	1,585,173	2,302,925	18,987,847	33,140,018	42,459	58,636	1,081,499	1,605,184

Graph Size (scalability). Figure 6 reports the size for varying number of nodes. As shown, TRC is much smaller than TR in all cases. For FB and FL, the largest SCC covers 83% and 70% of the graph respectively, while for YT, it covers just 45% (see Table 3). Thus, the TRC size for the FB dataset is 89% smaller, while for the YT and FL datasets, we achieve 40% and 57% of compression respectively. The larger the SCCs, the higher the compression achieved.

Since the size of the transitive closure (TC) grows rapidly, we compute TC for a smaller subset of the FB dataset varying the number of nodes from 1,000 to 6,000. As shown in Table 4, even for this small graph, the size of TC reaches 106 MB.

Percentage of Deletes. For each dataset, we vary the percentage of edge deletes from 0% to 30% of edge insertions. Table 5 presents the results for the FB dataset. We observe that the size of TR and TRC decreases; this can be explained by the fact that deletions affect the isolated nodes that become disconnected from the components and thus there are less edges between components and isolated nodes. The size of VG remains constant, since the size of the lifespan labels remains the same. Finally, the size of TRCH increases, because in case of deletes, additional nodes need to be included in the 2hop labels for ensuring the reachability test.

Table 4: Comparison with transitive closure

# nodes	Size (MB)			Constr. Time (sec)		
	TR	TRC	TC	TR	TRC	TC
1,000	0.013	0.012	2.91	0.01	4.76	167.49
2,000	0.026	0.009	11.56	0.23	5.02	1,457
3,000	0.039	0.012	26.27	0.35	5.89	5,788
4,000	0.052	0.018	47.12	0.41	6.33	16,580
5,000	0.063	0.026	73.97	0.59	6.79	39,112
6,000	0.074	0.032	106.82	0.72	7.13	81,123

Snapshot Granularity. Table 6 reports the storage required for maintaining daily, weekly and monthly snapshots of the three datasets. All sizes increase with the number of snapshots. For example, for FL, the increase of the number of snapshots by a factor of 30 (from 5 monthly to 134 daily) causes an increase of the size of TR by a factor of 3.44. The size of TR and TRC decreases with the snapshot granularity (number of snapshots) since less snapshots mean less postings and smaller SCC graphs. The size of VG does not decrease significantly, because it requires memory to keep lifespan labels for all nodes and edges of the graph.

Table 5: Size per % of deletes (Facebook)

% of deletes	Size (MB)			
	VG	TR	TRC	TRCH
0	11	0.5	0.21	1,493
10	11	0.58	0.22	1,528
20	11	0.45	0.19	1,612
30	11	0.47	0.18	1,664

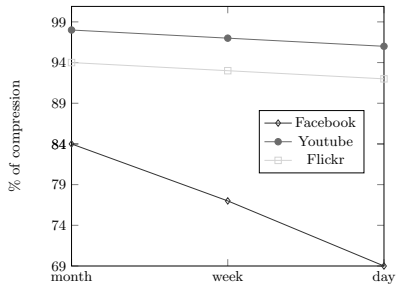


Figure 5: Compression ratio achieved by posting sharing

Posting Sharing. Finally, let us take a closer look at the posting sharing optimization by evaluating the reduction in the size of postings for various granularities as depicted in Figure 5. In general, we achieve compression ratios for the posting around 70% for FB, around 90% for FL and over 95% for YT. The compression ratio decreases with snapshot granularity due to the increase of the posting combinations. This is more evident for the FB dataset where the number of snapshots is higher.

6.2 Construction Time

In this set of experiments, we evaluate the time to construct the various indexes.

As seen in Figure 7, TRC is slower than TR, because of the additional time required for performing the bipartite matching. TRCH is even slower, since it also needs to construct the 2hop labels. We use the greedy algorithm for the bipartite matching and the INOUT strategy for computing the interval-2hop labels.

Constructing the TC for the whole graphs is prohibitive, since even for only $6,000$ nodes, it takes over 22 hours, while

Table 6: Size (MB) per snapshot granularity

	Facebook			*YouTube*			*Flickr*		
	Days	Weeks	Months	Days	Weeks	Months	Days	Weeks	Months
VG	11	6	5	7.87	7.34	6.94	45.52	39.85	38.15
TR	0.58	0.47	0.42	44.28	21.28	14.98	141	73	41
TRC	0.22	0.08	0.07	3.21	1.92	1.46	2.89	2.27	1.88
TRCH	1,528	1,041	845	5,865	4,936	4,062	7,951	6,684	5,719

the TR construction takes just 0.72 seconds (Table 4).

Comparison of Different Bipartite Matching Algorithms. We also constructed the TRC using the Hungarian algorithm. For all datasets, the size of the resulting TRC is almost equal to the size of the TRC resulting from using the greedy algorithm (the difference is in the order of KB), thus confirming our expectation that greedy achieves a very close approximation of the optimal solution for social graphs. The Hungarian algorithm is much slower than greedy requiring an additional 1.5 hour for large datasets such as FL.

Comparison with 2hop for insert only. We adopted the pruned labeling algorithm proposed in [2] for distance queries to create an indexing scheme for historical reachability queries. Pruned labeling incrementally updates the index for each newly inserted edge, whereas in our approach we compute 2hop labels per snapshot. The pruned labeling algorithm does not support deletions, thus, we compare the two algorithms on the Facebook dataset without deletions. The pruned algorithm was found to be 5.4 times faster but it produced labels that were 12 times larger that the ones computed with our approach.

6.3 Query Processing

Let us now focus on query processing. In each experiment, we ran 500 historical reachability queries where the source and target nodes are chosen uniformly at random with the restriction that both nodes are present in the graph at the beginning and the end of the query interval. Queries involving nodes not present either at the beginning or the end of the query interval can be pruned fast by checking the lifespans of the nodes.

Online Traversal of the Version Graph. Let us first compare between an instant-based (INS) and an interval-based (INT) online traversal of the version graph for different time intervals (Figures 8 and 9). A general remark that holds independently of the method used to evaluate queries is that false conjunctive queries are faster than true conjunctive queries, since processing stops as soon as a time instant is found at which the two nodes are not reachable. Analogously, true disjunctive queries are faster than false disjunctive queries, since processing stops as soon as a time instant is found at which the two nodes are reachable.

Interval-based traversal is faster that instant-based traversal for almost all datasets and query types, since it can find the answer faster by searching for longer intervals. The only exception is FB and false conjunctive queries, where INS is slightly better. This happens because with INS, the search stops as soon as the first false answer is produced in any traversal. Hence, if this answer is found in the first few time instants of the query interval negative answers can be produced quickly for the smaller graph (i.e, the FB graph).

Online Traversal versus TimeReach. Let us now compare interval-based online traversal with query processing using the TR, TRC and TRCH approaches. The results for conjunctive queries are shown in Figure 10 and for disjunctive queries in Figure 11.

We see that all approaches are not significantly affected by the increase of the query interval due to fast posting lookups and short distances in the SCC graph for the TR and TRC, and the efficient implementation of edge lifespans for the version graph. We see that the TRC approach does not only produce a smaller structure than TR but it also attains faster query response for almost all datasets. TR is slower because for answering a query it needs to traverse the SCC graph per time instant when the query nodes do not belong to the same component. TRCH attains the fastest time when compared with all other approaches. The performance of TRCH is expected, since only two simple steps are needed: first to obtain the intersection $L_{in}(v) \cap L_{out}(u)$, and after that to check the lifespans $\mathcal{L}$ of the nodes in the intersection.

7. CONCLUSIONS

Most real-life graphs evolve over time. In this paper, we address the problem of efficiently answering historical reachability queries over such graphs. Such queries ask whether a node u was reachable from another node v during a time interval in the past. We have proposed an approach termed *TimeReach* that exploits the fact that most graphs consist of strongly connected components (SCCs). TimeReach maintains information about SCC membership for each node, and a graph which represents the links between the strongly connected components. We also maintain a condensed version graph which corresponds to the version graph of the SCCs evolution. Our extensive experiments with three real social network datasets show that TimeReach is storage-efficient and can be constructed incrementally with a small overhead. Historical queries are processed efficiently even when involving large time intervals.

There are many possible directions for future work. One such direction is exploiting TimeReach towards answering other types of historical queries, such as shortest path ones. Another direction concerns the distribution of TimeReach. Distribution may either be based on time or exploit the SCC evolution by placing together nodes that belong to the same SCCs.

8. ACKNOWLEDGMENTS

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

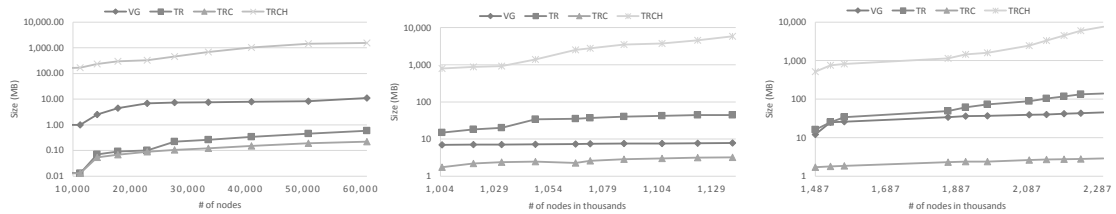


Figure 6: Size (log scale) for varying number of nodes in FB (left), YT (middle) and FL (right)

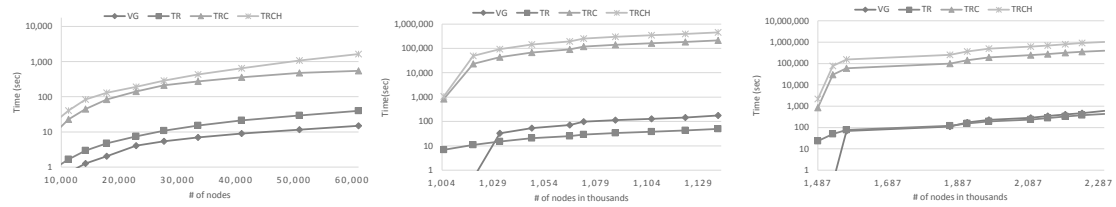


Figure 7: Construction time (log scale) for varying number of nodes in FB (left), YT (middle) and FL (right)

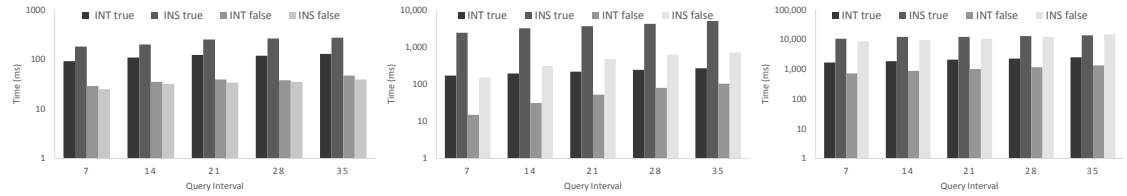


Figure 8: Query time (log scale) INS and INT for conjunctive queries in FB (left), YT (middle) and FL (right)

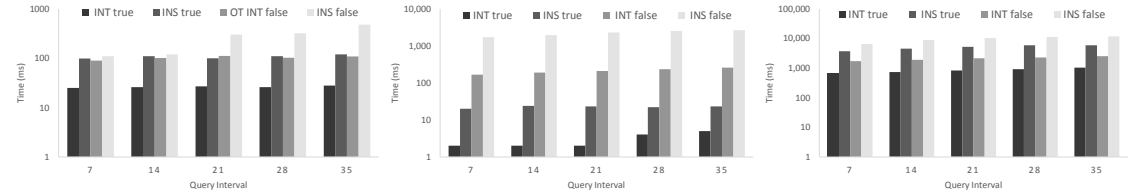


Figure 9: Query time (log scale) INS and INT for disjunctive queries in FB (left), YT (middle) and FL (right)

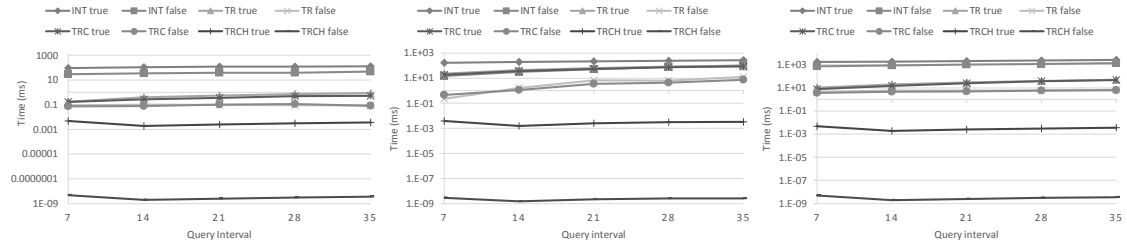


Figure 10: Query time (log scale) for conjunctive queries in FB (left), YT (middle) and FL (right)

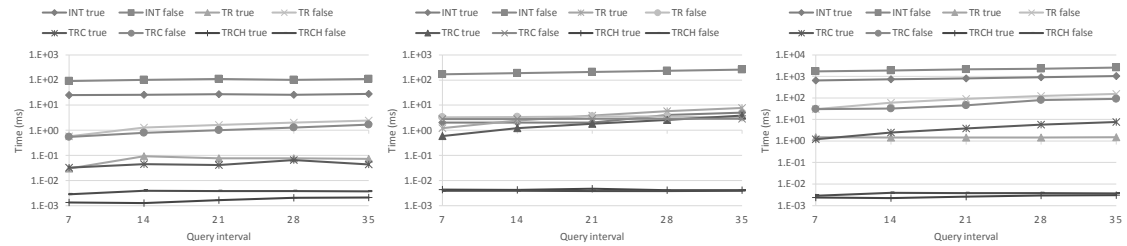


Figure 11: Query time (log scale) for disjunctive queries in FB (left), YT (middle) and FL (right)

9. REFERENCES

[1] Rakesh Agrawal, Alexander Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *SIGMOD*, pages 253–262, 1989.

[2] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *WWW*, pages 237–248, 2014.

[3] Ramadhana Bramandia, Byron Choi, and Wee Keong Ng. On incremental maintenance of 2-hop labeling of graphs. In *WWW*, pages 845–854, 2008.

[4] Li Chen, Amarnath Gupta, and M. Erdem Kurul. Stack-based algorithms for pattern matching on dags. In *VLDB*, pages 493–504, 2005.

[5] Yangjun Chen and Yibin Chen. An efficient algorithm for answering graph reachability queries. In *ICDE*, pages 893–902, 2008.

[6] Jiefeng Cheng, Jeffrey Xu Yu, Xuemin Lin, Haixun Wang, and Philip S. Yu. Fast computing reachability labelings for large graphs with high compression rate. In *EDBT*, pages 193–204, 2008.

[7] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003.

[8] Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for approximate and exact maximum weight matching. *Arxiv preprint arXiv:1112.0790*, 2011.

[9] Wenyu Huo and Vassilis J. Tsotras. Efficient temporal shortest path queries on evolving social graphs. In *Conference on Scientific and Statistical Database Management, SSDBM '14, Aalborg, Denmark, June 30 - July 02, 2014*, page 38, 2014.

[10] H. V. Jagadish. A compression technique to materialize transitive closure. *ACM Trans. Database Syst.*, 15(4):558–598, 1990.

[11] Christian S. Jensen and Richard T. Snodgrass. Temporal element. In *Encyclopedia of Database Systems*, page 2966. 2009.

[12] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. Efficiently answering reachability queries on very large directed graphs. In *SIGMOD*, pages 595–608, 2008.

[13] Udayan Khurana and Amol Deshpande. Efficient snapshot retrieval over historical graph data. In *ICDE*, pages 997–1008, 2013.

[14] Georgia Koloniari, Dimitris Souravlias, and Evaggelia Pitoura. On graph deltas for historical queries. *WOSS*, 2012.

[15] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *ACM SIGKDD*, pages 611–617, 2006.

[16] Alan G. Labouseur, Jeremy Birnbaum, Paul W. Olsen Jr, Sean R. Spillane, Jayadevan Vijayan, Jeong-Hyon Hwang, and Wook-Shin Han. The g* graph database: efficiently managing large distributed dynamic graphs. *Distributed and Parallel Databases*, To appear, 2014.

[17] Alan G. Labouseur, Paul W. Olsen, and Jeong-Hyon Hwang. Scalable and robust management of dynamic graph data. In *VLDB*, pages 43–48, 2013.

[18] Alan Mislove. Online social networks: Measurement, analysis, and applications to distributed information systems. Rice University, Department of Computer Science, 2009.

[19] Alan Mislove, Hema Swetha Koppula, Krishna P. Gummadi, and Bobby Bhattacharjee Peter Druschel. Growth of the flickr social network. In *ACM SIGCOMM WOSN*, pages 25–30, 2008.

[20] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *ACM SIGCOMM IMC*, pages 29–42, 2007.

[21] Chenghui Ren, Eric Lo, Ben Kao, Xinjie Zhu, and Reynold Cheng. On querying historical evolving graph sequences. *PVLDB*, 4(11):726–737, 2011.

[22] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Hopi: An efficient connection index for complex xml document collections. In *EDBT*, pages 237–255, 2004.

[23] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Efficient creation and incremental maintenance of the hopi index for complex xml document collections. In *ICDE*, pages 360–371, 2005.

[24] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[25] Silke Trißl and Ulf Leser. Fast and practical indexing and querying of very large graphs. In *SIGMOD*, pages 845–856, 2007.

[26] Sebastiaan J. van Schaik and Oege de Moor. A memory efficient reachability data structure through bit vector compression. In *SIGMOD Conference*, pages 913–924, 2011.

[27] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *ACM SIGCOMM WOSN*, pages 37–42, 2009.

[28] Haixun Wang, Hao He, Jun Yang, Philip S. Yu, and Jeffrey Xu Yu. Dual labeling: Answering graph reachability queries in constant time. In *ICDE*, page 75, 2006.

[29] Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *CIKM*, pages 1601–1606, 2013.

[30] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. Grail: a scalable index for reachability queries in very large graphs. *VLDB J.*, 21(4):509–534, 2012.

[31] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. Dagger: A scalable index for reachability queries in large dynamic graphs. *CoRR*, abs/1301.0977, 2013.

The VLDB Journal
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

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

Published online: 17 May 2013

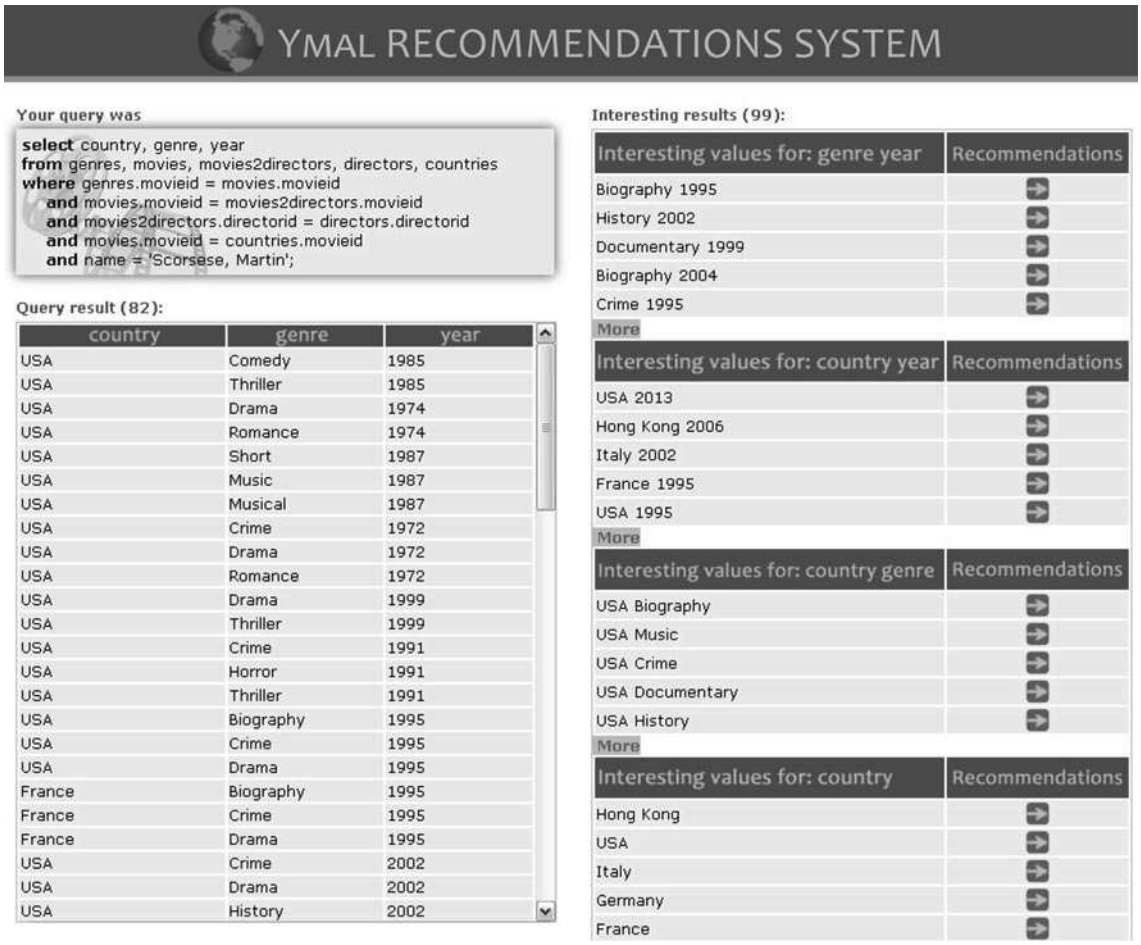


Fig. 1 YMALDB: On the *left side*, the original user query Q is shown at the *top* and its result at the *bottom*. Q asks for the countries, genres and years of movies directed by M. Scorsese. On the *right side*, interesting parts of the result are presented grouped based on their attributes and ranked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

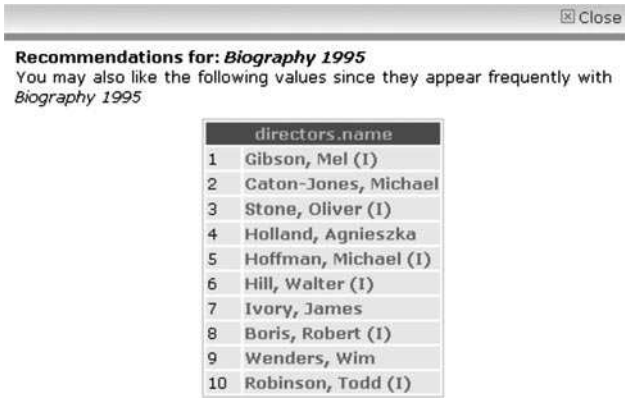


Fig. 2 YMALDB: Recommendations for a specific interesting piece of information (Biography films of 1995)

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.

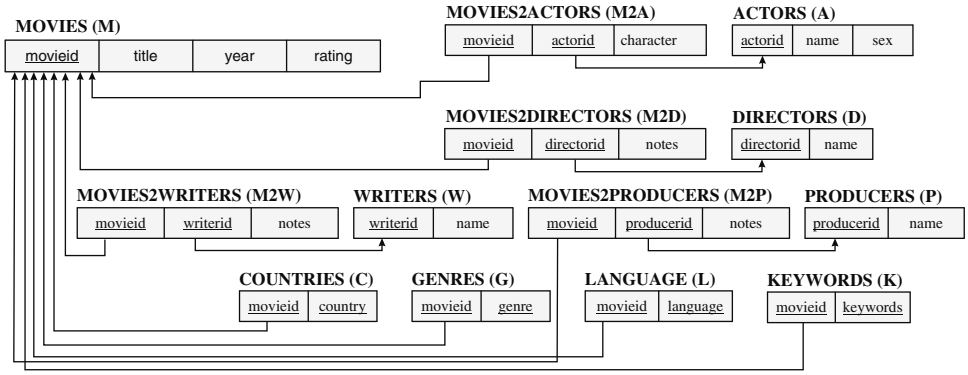
For a faSet f, we use $Att(f)$ to denote the set of attributes that appear in f. Let t be a tuple from a set of tuples S with schema R; we say that t satisfies a faSet f, where $Att(f) \subseteq R$, if $t[A_i] = a_i$, for all predicates $(A_i = a_i) \in f$ and $l_i \leq t[A_i] \leq u_i$, for all predicates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

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

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 $proj(\hat{Q}) = Att(scond(Q))$, $rel(\hat{Q}) = rel(Q) \cup \{R' | A_i \in R',$ 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 60 s, 10 movies produced in the 70 s and 20 movies produced in the 80 s. 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*

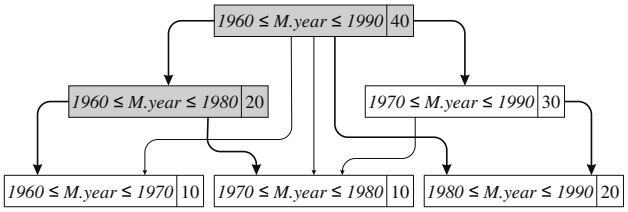


Fig. 6 Example of a minimum (m, ϵ)-cover set (depicted in gray) for the faSets depicted here ($m = 1$) along with their counts for $\epsilon = 1.0$. Arrows represent subsumption relations and bold arrows represent ϵ-subsumption relations

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 $|\epsilon_i\text{-CRFs}| > b$ and $|\epsilon_{i-1}\text{-CRFs}| \leq b$ 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 $|\epsilon_{i+1}\text{-CRFs}| = b$ or $|\epsilon_{i+1}\text{-CRFs}| = |\epsilon_i\text{-CRFs}|$.

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

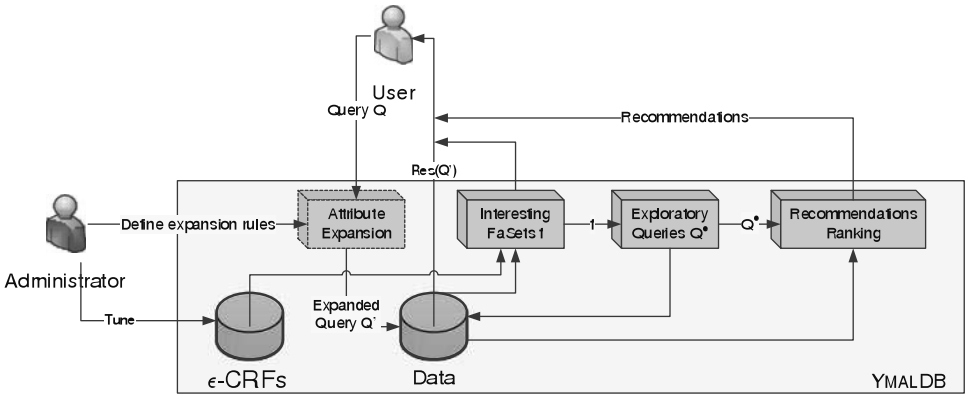


Fig. 7 The system architecture of YMALDB

$A^1 \rightarrow B^1, \ldots, A^r \rightarrow B^r$ 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

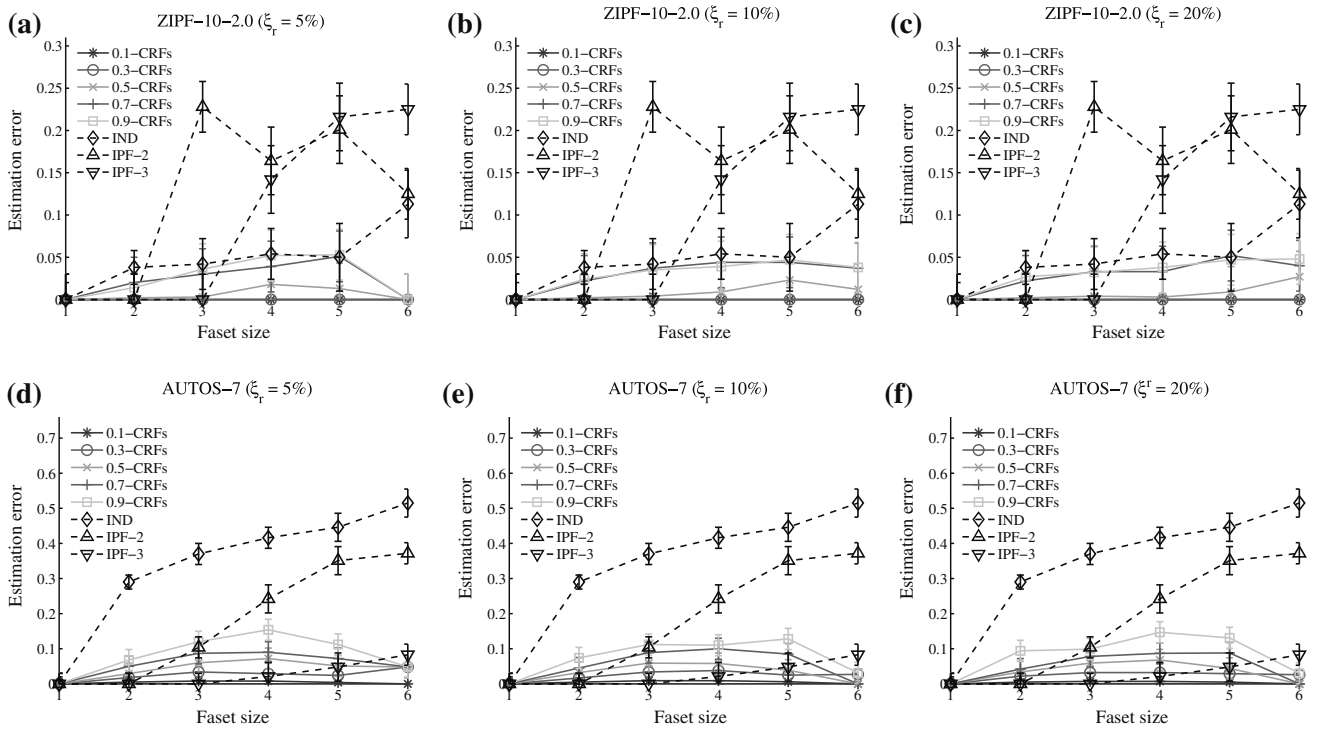


Fig. 8 Estimation error for 100 random rare faSets for different values of ξ_r when varying ϵ. **a** `ZIPF-10-2.0` ($\xi_r = 5\,\%$), **b** `ZIPF-10-2.0` ($\xi_r = 10\,\%$), **c** `ZIPF-10-2.0` ($\xi_r = 20\,\%$), **d** `AUTOS-7` ($\xi_r = 5\,\%$), **e** `AUTOS-7` ($\xi_r = 10\,\%$), **f** `AUTOS-7`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

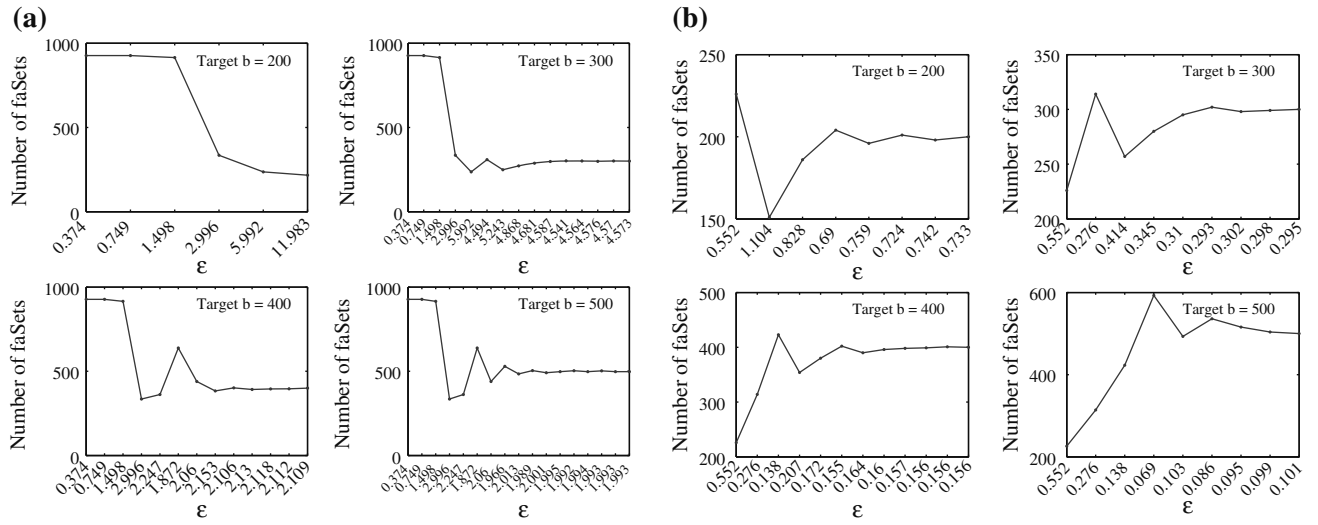


Fig. 9 Automatically suggesting values for ϵ given a storage limit b. **a** `ZIPF-7-1.0` ($\xi_r = 10\,\%$), **b** `AUTOS-7` ($\xi_r = 10\,\%$)

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

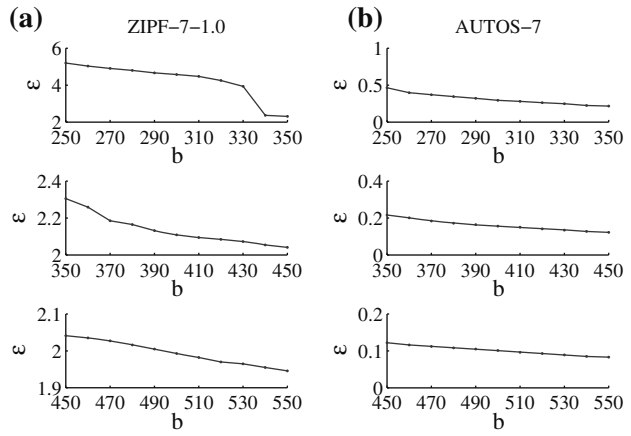


Fig. 10 Suggested ϵ values when varying b. **a** `ZIPF-7-1.0` ($\xi_r = 10\,\%$), **b** `AUTOS-7` ($\xi_r = 10\,\%$)

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

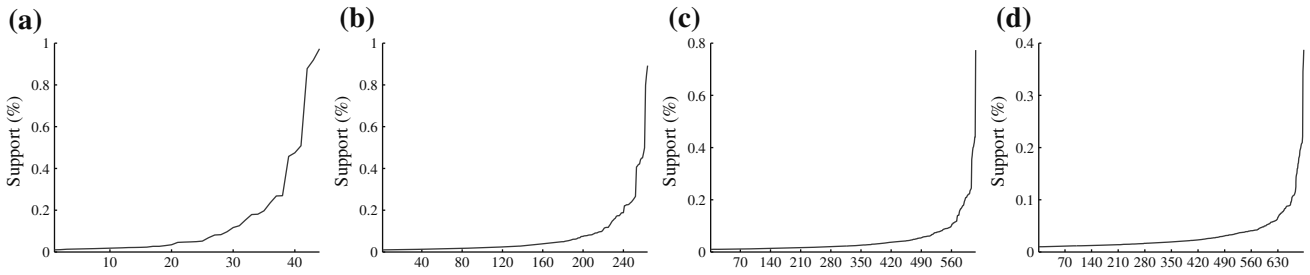


Fig. 11 Support of the faSets for the `AUTOS` dataset (x-axis is the number of faSet ordered by their support, e.g., x=50 means that this is the 50th less frequent faSet. **a** FaSet size 1, **b** FaSet size 2, **c** FaSet size 3, **d** FaSet size 4

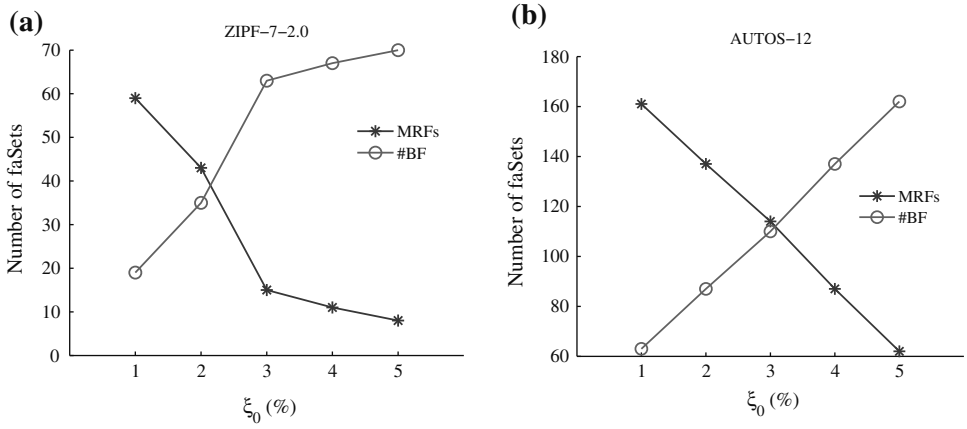


Fig. 12 Number of generated MRFs and number of faSets inserted into the Bloom filter for different values of ξ_0 when $\xi_r = 10\,\%$. **a** `ZIPF-7-2.0`, **b** `AUTOS-12`

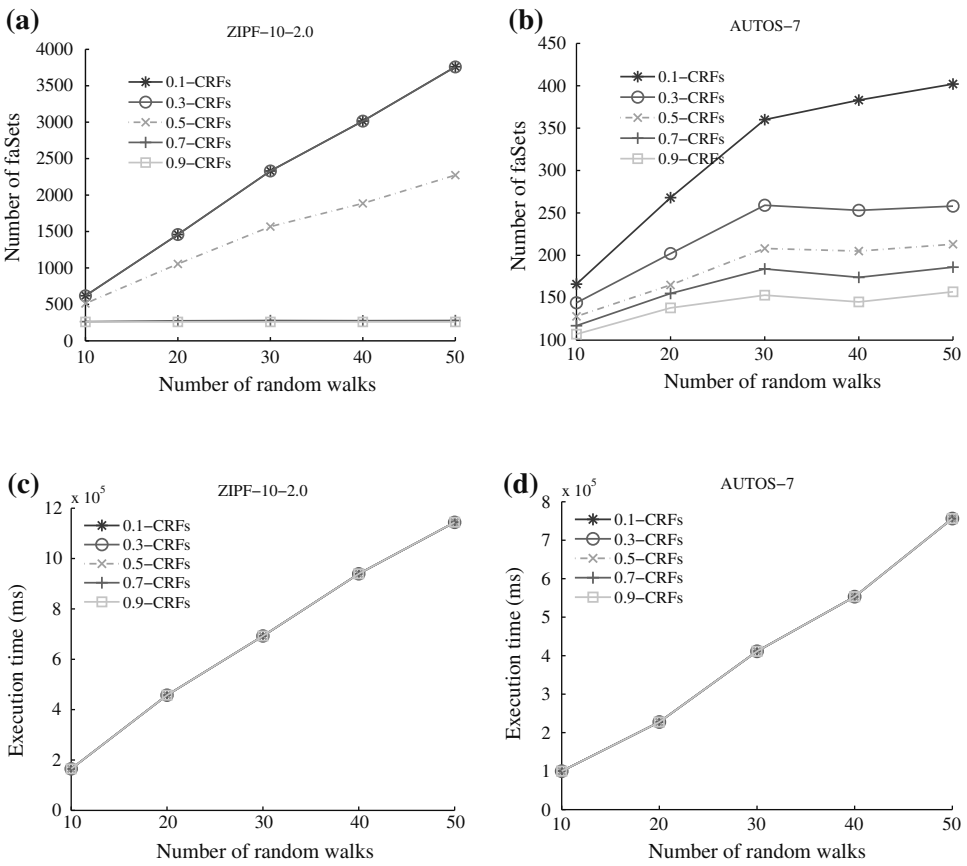


Fig. 13 Number of produced faSets (*top row*) and execution time (*bottom row*) when using different numbers of random walks for generating faSets for $\xi_r = 5\,\%$. **a** `ZIPF-10-2.0`, **b** `AUTOS-7`, **c** `ZIPF-10-2.0`, **d** `AUTOS-7`

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.

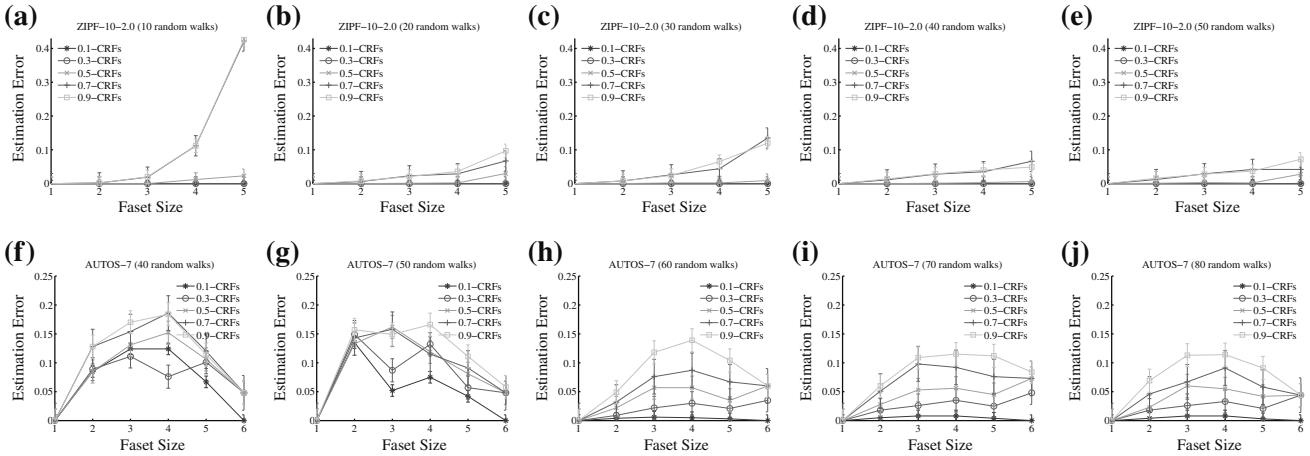


Fig. 14 Estimation error for 100 random rare faSets and $\xi_r = 5\,\%$ for different number of random walks employed during the generation of ϵ-CRFs when varying ϵ. **a** `ZIPF-10-2.0` (10 random walks), **b** `ZIPF-10-2.0` (20 random walks), **c** `ZIPF-10-2.0` (30 random walks), **d** `ZIPF-10-2.0` (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)

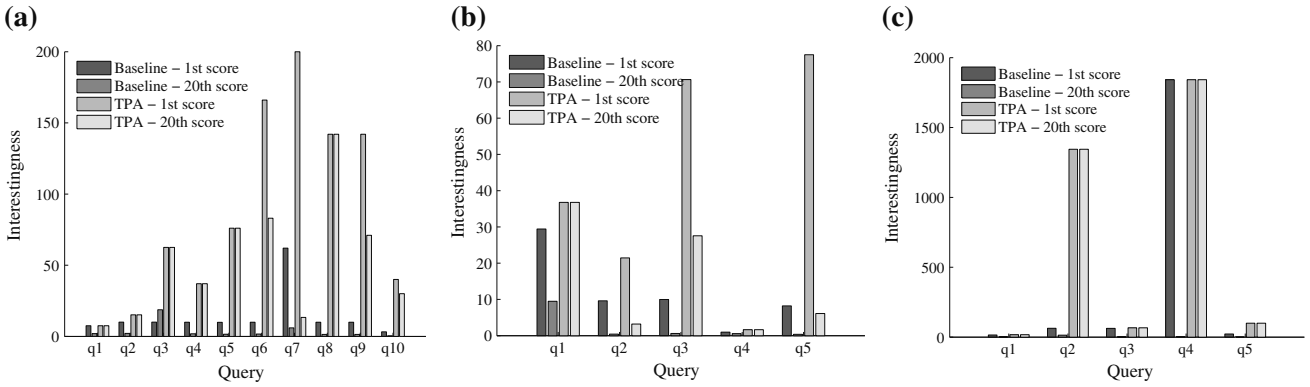


Fig. 15 Interestingness scores of the top 20 most interesting faSets retrieved by the TPA and the baseline approach. **a** `ZIPF-10-2.0`, **b** `AUTOS-12`, **c** `MOVIES`

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.

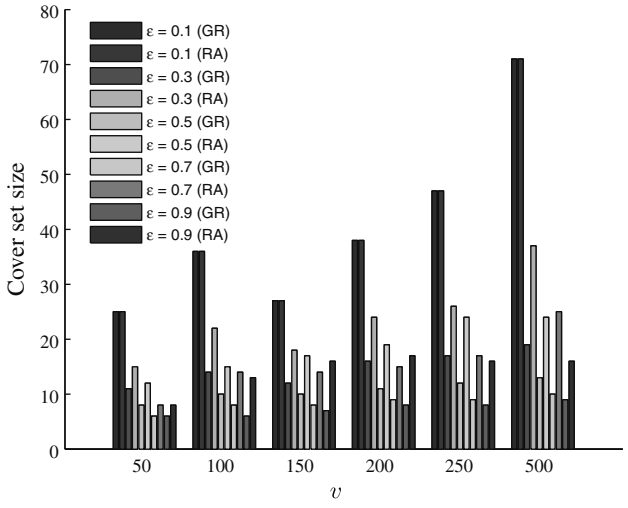


Fig. 16 Size of the produced cover sets by the greedy heuristic (GR) and the random approach (RA) when varying the number of initial faSets v

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*"}

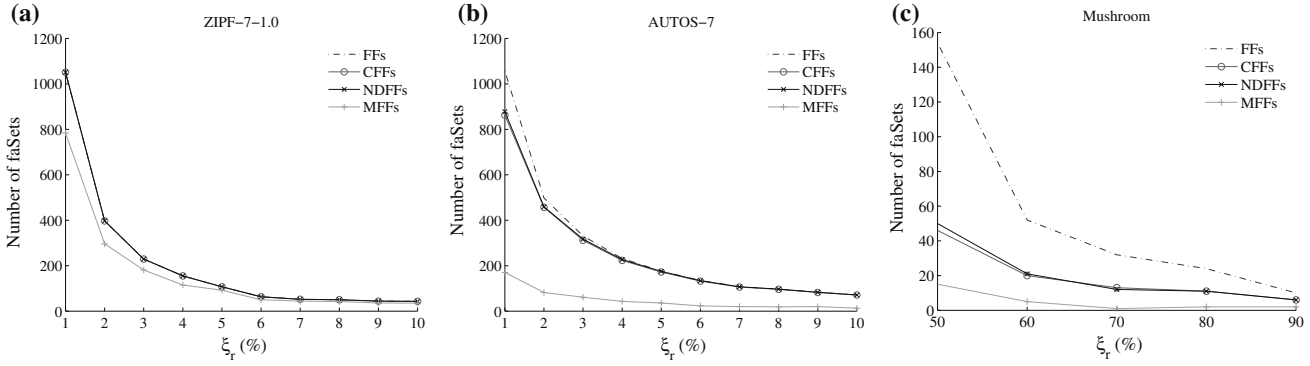


Fig. 19 Frequent, closed frequent, non-derivable frequent and minimal frequent faSets for various datasets. **a** `ZIPF-7-1.0`, **b** `AUTOS-7`, **c** `Mushroom`

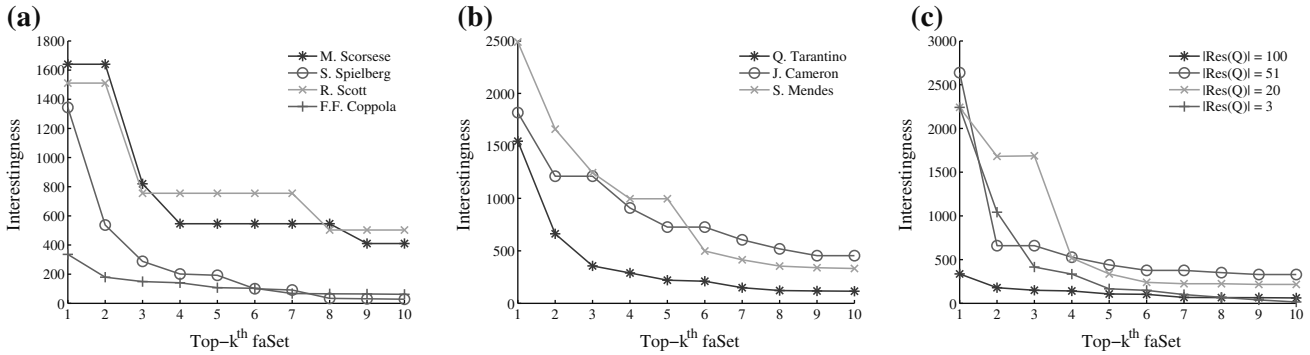


Fig. 20 Interestingness of the top-10 faSets for queries with different number of results. **a** Different queries ($|Res(Q)| \approx 100$), **b** Different queries ($|Res(Q)| \approx 30$), **c** Different ($|Res(Q)|$)

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)

Information Systems 53 (2015) 60–86

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

CineCubes: Aiding data workers gain insights from OLAP queries

Dimitrios Gkesoulis [a,1], Panos Vassiliadis [b,*], Petros Manousis [b]

[a] UTC Creative Lab, Ioannina, Hellas
[b] University of Ioannina, Ioannina, Hellas

ARTICLE INFO

Article history:
Received 10 March 2014
Received in revised form
22 October 2014
Accepted 17 December 2014
Available online 3 January 2015

Keywords:
CineCubes
Data narration
Data movies
Insight generation
OLAP
Management of query results
Query recommendation

ABSTRACT

In this paper we demonstrate that it is possible to enrich query answering with a short data movie that gives insights to the original results of an OLAP query. Our method, implemented in an actual system, *CineCubes*, includes the following steps. The user submits a query over an underlying star schema. Taking this query as input, the system comes up with a set of queries complementing the information content of the original query, and executes them. For each of the query results, we execute a set of *highlight* extraction algorithms that identify interesting patterns and values in the data of the results. Then, the system visualizes the query results and accompanies this presentation with a text commenting on the result highlights. Moreover, via a text-to-speech conversion the system automatically produces audio for the constructed text. Each combination of visualization, text and audio practically constitutes a movie, which is wrapped as a PowerPoint presentation and returned to the user.

1. Introduction

Can we answer user queries with data movies? Why should query results be treated simply as sets of tuples returned by the DBMS as if they would be visualized in an orange CRT of the 70s? So far, database systems assume their work is done once results are produced, effectively prohibiting even well-educated end-users to work with them. Can we do something better?

In this paper, we revise the traditional assumptions of *query answering* in order to raise the issue of *insight gaining*. We serve the purpose of insight gaining in two ways, by demonstrating that

- *it is possible to produce query results that are* (*a*) *properly visualized*, (*b*) *textually exploitable*, *i.e.*, *enriched with an automatically extracted text that comments on the result*, (*c*) *vocally enriched*, *i.e.*, *enriched with audio that allows the user not only to see*, *but also hear*, and,
- *it is possible to come up with a working*, *extensible method that accompanies a query result with the results of complementary queries which allow the user to contextualize and analyze the information content of the original query.*

Interestingly, *an insightful sequence of related queries that provide context and depth to the original query*, "*dressed*" *with the appropriate visualization and sound*, *ends up to be nothing else but a data movie where cubes star*.

Motivation: Yet, what does *insight* mean? In a recent approach, Dove and Jones [1] combine the definitions from the communities of Information Visualization and Cognitive Psychology: whereas the InfoVis community defines insight as "*something that is gained*" (after the observation of data by a participant), psychologists define it as an "*Aha*!" *moment which is experienced*. Interestingly, the two definitions can be

* Corresponding author.
E-mail address: pvassil@cs.uoi.gr (P. Vassiliadis).
[1] Work conducted while in the Univ. of Ioannina.

http://dx.doi.org/10.1016/j.is.2014.12.006
0306-4379/

combined in a common view, where once the user works with information, starting with an original state of mind on the current state of affairs, there is an "Aha!" moment, where the user suddenly realizes a new way of looking at the data, resulting in a new mental model for the state of affairs, or else, new understanding [1].

In order to facilitate the "Aha!" moment that creates insight, the scientific community is spending more and more effort nowadays in the area of data analysis. In a recent SIGMOD keynote speech in 2012 [2], Pat Hanrahan from Stanford University and Tableau Software makes a case for visual analytics as the best way to support the data analysis process; whereas the former involves the automatic extraction of information accompanied by the appropriate visualization, the latter can be summarized as follows: "get the data, deliver them in a clean usable form, contextualize them, extract relationships and patterns hidden within them, generalize for insight, confirm hypotheses and errors, share with others, decide and act".

Our goal with the CineCubes system is to provide an extensible tool that acts as the platform that supports the insight generation of the data analysis lifecycle (contextualization, pattern extraction, insight) by producing small stories that make an impression "in a memorable way" to the data enthusiast or the data worker who performs data analysis.

Key contributions: Then, the question arises: *And how can we do that?* In a nutshell, our main result is the introduction of a fully automated and extensible method that allows the generation of a data movie, over an *OLAP* database, with a simple user query as starting point. In detail, our individual assumptions and contributions can be listed as follows:

- We start with a realistic assumption that empowers us with the ability to address the challenge in a clear setting. We assume the existence of a star schema with clean, reconciled hierarchies of reference data; we also assume that the end users are interested in working with OLAP queries over these data.
- We demonstrate how to complement the original query with additional queries that allow the contextualization and analysis of the original result. To provide contextualization, we exploit the defining values (*i.e.*, selection conditions) of the original query and automatically generate complementary queries that compare its results with the results of queries having similar values. Practically, this exploits OLAP hierarchies and compares sibling values within the same hierarchy. For example, if the user has scoped the data of interest with selection conditions *Continent*=North America and *Gender*=Male, we accompany the original query with queries that compare North America to other continents and men to women. To provide further analysis of the results, we drill in the grouping levels of the original result to see the breakdown of its (aggregate) measures and understand its internal structure. So, for example, if the user originally aggregates information by 10 *Year* intervals and *Continent*, we provide details by drilling-in to 5 *Year* intervals and *Country*.
- Whereas the above actions produce a first step towards supporting the contextualization and analysis of the data, we have also implemented a fully automated mechanism for producing patterns and trends within each of the above results. To this end, we introduce *highlight extraction methods* that operate on the result of a query and discover interesting findings (like e.g., the fact that a column contains a large share of the highest or lowest values of a result, or that a row systematically has higher/lower values than another). These highlights serve also the visual presentation of the data via appropriate coloring of important values.
- Automatic highlight generation is a key contribution and not only for visualization purposes. In this paper, we also demonstrate how to automate the generation of text describing the aforementioned highlight findings (by accompanying each type of highlight with a template text) and how to convert this text to audio (via publicly available text-to-speech conversion software).
- Much like movies, we organize our stories in acts, with each act including several episodes all serving the same purpose. We demonstrate that all the above can be packaged with small programming effort in a PowerPoint presentation, practically presenting a small movie to the user. The emphasis on small programmatic effort is intended: an important goal of this paper is to demonstrate that the technical barrier for someone who would be interested to conduct research on this problem is low. Existing API's for the construction of PowerPoint presentations [3] and for text to speech conversion [4] allow us to produce a PowerPoint presentation programmatically: each query can have a slide where its result is neatly visualized; the slide's notes can contain the text explaining the result and the slide's audio can be produced via text-to-speech conversion.
- Quite importantly, and orthogonally to the above, we have intentionally built our system in a way that is both fully automated and extensible. Extensibility has been a cornerstone of our approach and it is best demonstrated by two extensibility mechanisms, (i) one concerning the generation of the complementary queries to the original question, and, (ii) another concerning the automatic identification of interesting highlights within the results of each query. In this paper, we discuss the points of extensibility of CineCubes in detail.

Target audience and added value: *Who can benefit from CineCubes?* There are many kinds of people that currently work with data in order to deliver a report, a live or a self-running presentation, an on-line talk, or a journal article, and who would all benefit from such a system. Business users with particular questions in mind are a first such case [2]. People creating self-running presentations [5] (*i.e.*, presentations publicly available for mass audiences without the presenter being involved), either in a film-clip or a slide-show can benefit from a system giving both insights and visual representations. Collaborative work in small groups [5] can benefit from CineCube presentations as they provide the basis to broaden the scope of the original search and lead to new questions to be answered. Journalists nowadays are more

and more preparing data-driven articles that involve working with data and using infographics to make a case (see [6] for a large list of examples – typically, New York Times, Washington Post and the Guardian are reference news media for infographics).

Overall, we claim that *any data worker creating a report summarizing findings and insights based on data can benefit from CineCubes* in many ways: automated highlight extraction, auxiliary query results, automatically generated text, audio and visual highlight do not only work together to generate contextualization, analysis, probes for further exploration, and ultimately, insight, but also provide a reusable means of precanned text and visual graphics that can speed-up the compilation of the desired report. The results of a user study that we have conducted (Section 5.4) reveal that improvements come in two ways, and specifically, (a) better quality and (b) faster creation of the report.

Novelty: We believe that despite the vast amount of work (refer to Section 6) in the areas of data visualization, query recommendation, pattern mining, and, to a lesser extent, text generation from query results, this paper makes a disruptive contribution *by raising the issue of gaining insight from the data via small data movies (as opposed to traditional, simple query answering) and providing an automated solution to it via (a) auxiliary queries and (b) automated highlight extraction.* The idea of a data movie has been a driver for the compilation of individual techniques in a single, fully automated and extensible packaging: a movie requires episodes in its structure, visual effects and audio; these needs have produced the solution of query sequences, automatic highlight extraction, as well as automatic text and audio generation via simple programmatic APIs. Thus, one should not regard the individual parts of the method as the novel contribution of the paper; it is their principled and extensible bundling in a single, extensible tool that creates a research opportunity and practically new research ground to explore.

Roadmap: In Section 2, we give an overview of the method as well as a reference example. In Section 3, we discuss our method's internals. In Section 4 we present the software architecture of CineCubes with special emphasis on the extensibility aspect and explain the low technical barrier of the method, too. In Section 5, we show experimental results in terms of efficiency and usability. In Section 6, we discuss related work. We conclude with a presentation of open issues in Section 7.

2. Method overview

2.1. Constructing a CineCube story

A really useful characteristic of cubes is that *dimensions provide a context for facts* [7]. This is especially important if combined with the fact that dimension values come in *hierarchies*; therefore, every single fact can be simultaneously placed in multiple hierarchically structured contexts, providing thus the ability to analyze sets of facts from multiple perspectives. For the reader who is not knowledgable of OLAP hierarchies, Fig. 2 depicts two dimension hierarchies in terms of both data and schema; for the latter, see for example, dimension *Education* which is characterized by a hierarchy of *dimension levels* $L_0 \to L_1 \cdots \to L_k$ (from the most detailed to the most coarse abstraction of values). At the same time, hierarchies allow the comparison of their members with (a) *ancestors*, (b) *descendants* and (c) *siblings* (children of the same parent). For example, in Fig. 2, the ancestor of value 'PhD' (defined at level L_1) at level L_2 is 'Post secondary' (denoted as $anc_{L_1}^{L_2}('PhD') = Post\text{-}Secondary$). Assume now, a detailed cube $\mathbf{DS}^0$ (which is the most detailed data set that we can work with – a.k.a a *fact table* in OLAP terminology). We require that $\mathbf{DS}^0$ is defined (a) over a set of dimensions $D = \{D_1, \ldots, D_n\}$ and (b) over a measure *M*.

A query *q* in our context exploits the multidimensionality of the cube space and can be considered as a quadruple $q = (\mathbf{DS}^0, \phi, \bar{L}, agg(M))$ where:

- ϕ is a conjunction of dimensional restrictions of the form $D_i L_j = value_i$ – i.e., constraints that focus the context of the query to certain dimensional values.
- $\bar{L}$ is a set of grouper dimensional levels (practically comprising the GROUP BY attribute set in a SQL query), over which the information will ultimately be grouped.
- *agg(M)* is an aggregate function applied to the measure of the cube; again, we restrict ourselves to a single measure.

The semantics of a cube query *q* in terms of SQL over a star schema defined by and *D* and $\mathbf{DS}^0$:

SELECT $\bar{L}$, $agg(M)$
FROM DS^0, $\mathbf{D}^q$
WHERE ϕ AND $\phi_{\bowtie}$
GROUP BY $\bar{L}$

where $\mathbf{D}^q$ and $\phi_{\bowtie}$ serve the syntactic correctness of the query, as $\mathbf{D}^q$ is the set of all the necessary dimension tables D_i that appear in the body of the query, $\mathbf{D}^q \subseteq \{D_1, \ldots, D_n\}$ and the expression $\phi_{\bowtie}$ is a conjunction of atoms needed to join DS^0 and the tables of $\mathbf{D}^q$. For practical reasons, in all our deliberations, the set $\bar{L}$ is restricted to two groupers $\{L_\alpha, L_\beta\}$, so that the results can be represented as a crosstab table in a 2D screen.

Given a query *q* and its result *q.RS*, we can create a short "data story" by seeking for answers to the following questions:

0. *A first assessment of the current state of affairs.* Practically, this requirement refers to the execution of the original query.
1. *Put the state in Context.* Are the results of *agg(M)* good? What does "good" mean in this case? Typically, we would expect to compare the result of the query *q* to the results of similar queries over siblings of the values that appear in the filter list ϕ.
2. *Analysis of why things are this way.* Given a certain cuboid that is the result of a query, we would like to provide some more insight on the presented results; one way to achieve this is to show the breakdown of the contributions of the detailed values to the overall, aggregate value. Practically speaking, this involves drilling-down for each of the involved groupers and presenting the analysis of the internal breakdown for each of the groupers.

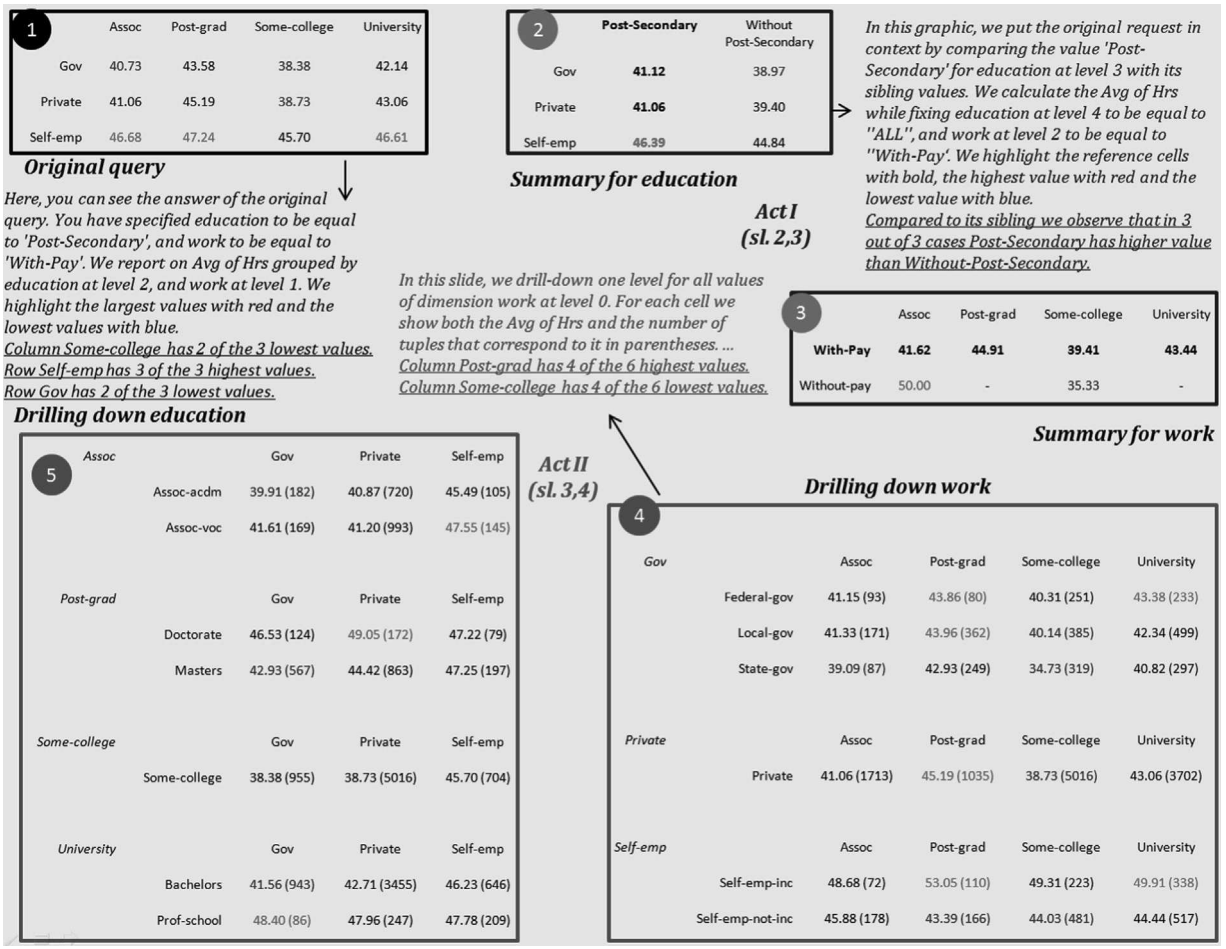


Fig. 1. An excerpt of a CineCubes story over the Adult data set. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Clearly, this set of complementary queries that a story comprises is extensible; existing and novel results in query recommendation (see Section 6) can be progressively plugged in our method in order to produce more informative CineCube movies.

2.2. Running example

To demonstrate our approach we use an example from the well known Adult (a.k.a census income) data set referring to data from 1994 USA census. There are 7 dimensions (*Age*, *Native Country*, *Education*, *Occupation*, *Marital status*, *Work class*, and *Race*) in the data set and a single measure, *Hours per Week*. We will use a uniform terminology to refer to the dimensions' levels, (L_0, L_1, ..). Also, the ragged dimensions are complemented with values identical to their parent, to make them balanced and fit to the model of [8].

We start with an original query where the user has fixed *Education* to 'Post-Secondary' (at level L_3), and *Work* to 'With-Pay' (at level L_2) and requests the *Avg* of *HrsPerWeek* grouped by *Education* at level 2, and *Work* at level 1. We depict these two dimensions in Fig. 2. We arrange the presentation of the result in columns (*Education*) and rows (*Work*). In Fig. 1, in the slide with the indication ❶ , one can also see the actual presentation as a 2D matrix, the visualization interventions (highlighting high and low values with color) and the text accompanying the visual presentation. The text is (a) part of the slide's notes (so that the user can reuse it) and (b) orally voiced as an audio file accompanying the slide. The slide's text is delivered via a set of *highlight extraction* methods that search the 2D matrix for prominent features (high and low values, rows or columns dominating some of these indicatory values, etc.).

Once the original query has been answered, we move on to put it in context. *Act I* of the CineCube movie, including slides ❷ and ❸ (dressed in blue color), performs the following analysis: since there is a selection condition with two atoms (*Education.L3*='Post-Secondary' and *Work.L2*='With-Pay'), we compare each of the defining values with its sibling. So, slide ❷ presents a comparison between the siblings of 'Post-Secondary' at level L_3 of *Education* (specifically, the single value 'W/O post secondary'). The analysis shows that in 3 out of 3 cases people with

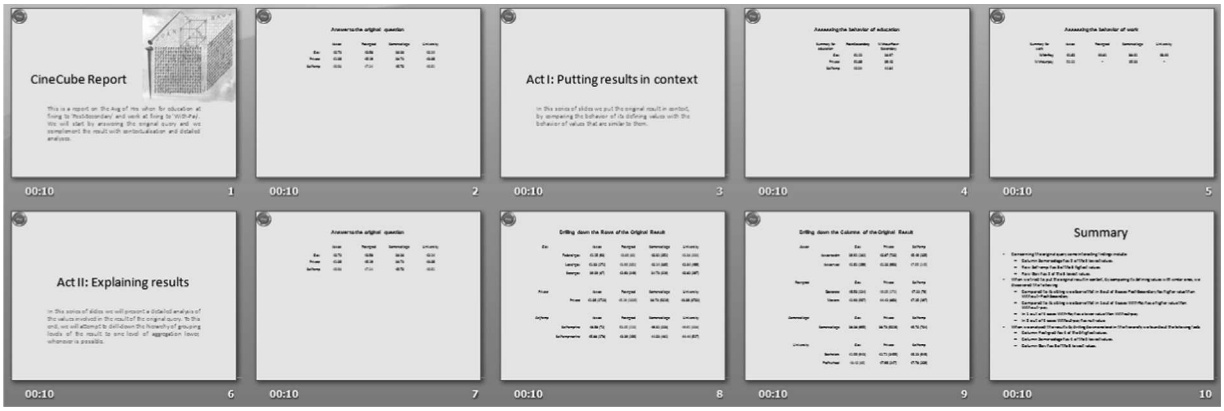


Fig. 3. A snapshot of the internal structure of the CineCube movie.

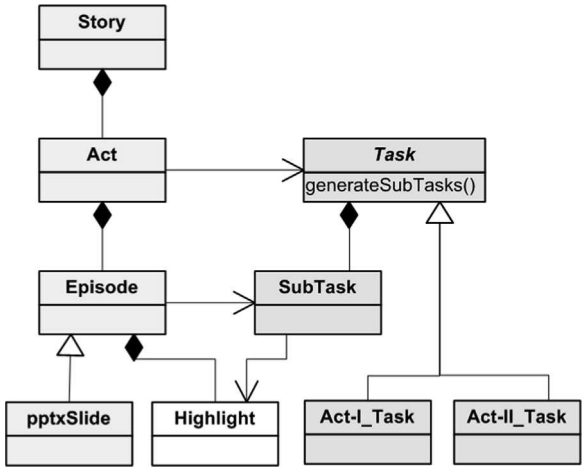


Fig. 4. Extensibility mechanism for CineCubes.

structure, with a set of classes that actually get the job done (right-hand side of Fig. 4). Specifically, the generation of queries (and slides) within each *Act* is delegated to the abstract class *Task*. For reasons of extensibility, *Task* is an abstract class: therefore, we materialize it differently for each kind of *Act* (in Fig. 4 we depict two such materializations, for Act I and Act II). The crux of the approach is that each episode comes with (typically one, but sometimes more) queries in its background; therefore, each Act generates *SubTasks*, with each *Subtask* carrying and being responsible for the execution of a query that gathers the data (that are ultimately visualized in the main part of the slide). An *Episode* can have several *SubTasks* to compute its contents. Since each *SubTask* carries its own query depending on the *Act*/*Task*, the above mechanism is extensible by appropriately constructing the method *generateSubTasks*() for each materialization of *Act*.

Moreover, the determination of key findings, or *Highlights* within each *Episode* is performed by the homonymous class. We fundamentally consider the presentation of results as a 2D matrix on the screen[2]; to this end, we have structured several methods that scan a 2D matrix and isolate interesting cells (top-k max or top-k min values, domination of a class of values by a column or row, etc.). The class *Highlight* is a point of extensibility where methods for result extraction can be added to search for more results within the answer of a query.

For more information on the internal structuring of CineCubes, we refer the interested reader to Section 4, where we discuss the software architecture as well as the two aforementioned *extensibility mechanisms* in more detail. Before that, however, our next step is to present the essence of our method along with its formal foundations.

3. Foundations and method internals

In this section, we start with a short description of the model for cubes and cube queries and then we move on to describe (a) acts, as the means for collecting data via complementary queries and (b) highlights as the means for automatically detecting some important findings within query results and the means for text construction. We also provide the basic steps of our method for the creation of CineCube movies.

3.1. Formal background

We base our approach on an OLAP model that involves (a) *dimensions* defined as lattices of dimension *levels*, (b) *ancestor functions*, mapping values between related levels of a dimension, (c) *detailed data sets*, practically modeling fact tables at the lowest granule of information for all their dimensions, and (d) *cubes*, defined as aggregations over detailed data sets. We follow the logical cube model of [8], accurately summarized in [11], which we customize here for the context of Cinecubes. For the reader who is knowledgable of the OLAP terminology but does not want to spend time on the formalities it is sufficient to refer to Section 2.1 for the intuition of the basic concepts; then, this subsection can be omitted.

Domains: We assume four countable pairwise disjoint infinite sets exist: a set of *level names* (or simply *levels*) $\mathcal{U}_{\mathcal{L}}$, a set of *measure* names (or simply *measures*) $\mathcal{U}_{\mathcal{M}}$, a set of *dimension* names (or simply *dimensions*) $\mathcal{U}_{\mathcal{D}}$ and a set of *cube* names (or simply *cubes*) $\mathcal{U}_{\mathcal{C}}$. The set of *attributes* $\mathcal{U}$ is defined as $\mathcal{U}=\mathcal{U}_{\mathcal{L}} \cup \mathcal{U}_{\mathcal{M}}$. For each $A \in \mathcal{U}_{\mathcal{L}}$, we define a

[2] Of course, other forms of visualization can *accompany* the result; however, it is our conviction that *the actual data should definitely be part of the answer* [10].

countable totally ordered set $dom(A)$, the domain of A, which is isomorphic to the integers. Similarly, for each $A \in \mathcal{U}_{M}$, we define an infinite set $dom(A)$, the domain of A, which is isomorphic to the real numbers. We can impose the usual comparison operators to all the values participating to totally ordered domains $\{<, >, \leq, \geq\}$.

Dimensions and levels: A *dimension* D is a lattice $(\mathbf{L}, \prec)$ such that:

- $\mathbf{L} = \{L_1, \dots, L_n\}$ is a finite subset of $\mathcal{U}_{\mathcal{L}}$.
- $dom(L_i) \cap dom(L_j) = \emptyset$ for every $i \neq j$.
- $\prec$ is a partial order defined among the levels of $\mathbf{L}$.
- The highest level of the hierarchy is the level *D. ALL* with a domain of a single value, namely 'D. all'.

Each path in the dimension lattice, beginning from its upper bound and ending in its lower bound is called a *dimension path*.

A family of functions $anc_{L_1}^{L_2}$ is defined, satisfying the following conditions:

1. For each pair of levels L_1 and L_2 such that $L_1 \prec L_2$, the function $anc_{L_1}^{L_2}$ maps each element of $dom(L_1)$ to an element of $dom(L_2)$.
2. Given levels L_1, L_2 and L_3 such that $L_1 \prec L_2 \prec L_3$, the function $anc_{L_1}^{L_3}$ equals to the composition $anc_{L_1}^{L_2} \circ anc_{L_2}^{L_3}$. This implies that:
   - $anc_{L_1}^{L_1}(x) = x$;
   - if $y = anc_{L_1}^{L_2}(x)$ and $z = anc_{L_2}^{L_3}(y)$, then $z = anc_{L_1}^{L_3}(x)$;
   - for each pair of levels L_1 and L_2 such that $L_1 \prec L_2$, the function $anc_{L_1}^{L_2}$ is monotone (preserves the ordering of values). In other words: $\forall x, y \in dom(L_1): x < y \Rightarrow anc_{L_1}^{L_2}(x) \leq anc_{L_1}^{L_2}(y), L_1 \prec L_2$.

Schemata and data sets: A *schema* $\mathbf{S}$ is a finite subset of $\mathcal{U}$. Normally, we will represent a schema as divided in two parts: $\mathbf{S} = [D_1 \cdot L_1, \dots, D_n \cdot L_n, A_1, \dots, A_m]$, where:

- $[L_1, \dots, L_n]$ are levels from a dimension set $\mathbf{D} = [D_1, \dots, D_n]$ and level L_i comes from dimension D_i, for $1 \leq i \leq n$.
- $[A_1, \dots, A_m]$ are attributes, i.e. measures and levels.

A *detailed schema* $\mathbf{S}^0$ is a schema whose levels are the lowest in the respective dimensions. When we refer to a level L as the lowest in the dimension, it means that there does not exist any other level L', such that $L' \prec L$.

A *tuple* t over a schema $\mathbf{S} = [D_1.L_1, \dots, D_n.L_n, A_1, \dots, A_m]$ is a total and injective mapping from $\mathbf{S}$ to $dom(L_1) \times \dots \times dom(L_n) \times dom(A_1) \times \dots \times dom(A_m)$, such that $t[X] \in dom(X)$ for each $X \in \mathbf{S}$.

A *data set* $\mathbf{DS}$ over a schema $\mathbf{S} = [D_1.L_1, \dots, D_n.L_n, A_1, \dots, A_m]$ is a finite set of tuples over S such that:

- $\forall t_1, t_2 \in \mathbf{DS}, t_1[L_1, \dots, L_n] = t_2[L_1, \dots, L_n] \Rightarrow t_1 = t_2$.
- for no strict subset $X \subset [L_1, \dots, L_n]$, the previous also holds.

In other words, $A_1, \dots, A_m$ are functionally dependent (in the relational sense) on levels $[L_1, \dots, L_n]$ of schema $\mathbf{S}$. A *detailed data set* $\mathbf{DS}^0$ is a data set over a detailed schema $\mathbf{S}^0$.

A *star schema* $(\mathbf{D}, \mathbf{S}^0)$ is a couple comprising a finite set of dimensions $\mathbf{D}$ and a detailed schema $(\mathbf{S}^0)$ defined over (a subset of) these dimensions.

Selection filters: An *atom* is *true*, *false*, (with obvious semantics) or an expression of the form $anc_{L_0}^{L_1}(L_0) = v$, or in shorthand, $L_1 = v$, with $v \in dom(L_1)$.

A *selection condition* ϕ is a formula involving atoms connected via the logical connective $\wedge$, with at most one atom per dimension. The expression $\phi(\mathbf{DS})$ is a set of tuples $\mathbf{X}$ belonging to $\mathbf{DS}$ such that when, for all the occurrences of level names in ϕ, we substitute the respective level values of every $x \in \mathbf{X}$, the formula ϕ becomes true. A *detailed selection condition* ϕ^0 is a selection condition where all participating levels are the detailed levels of their dimensions.

Cube queries: The user can submit *cube queries* to the system. A cube query specifies (a) the (basic) cube over which it is imposed, (b) the selection condition that isolates the records that qualify for further processing, (c) the aggregator levels, that determine the level of coarseness for the result, and (d) an aggregation over one of the measures of the underlying cube that accompanies the aggregator levels in the final result. More formally, a *cube query* is an expression of the form:

$$q = (\mathbf{DS}^0, \phi, [L_\alpha, L_\beta], agg(M_m^0))$$

where:

- $\mathbf{DS}^0$ is a detailed data set over the schema $\mathbf{S} = [L_1^0, \dots, L_n^0, M_1^0, \dots, M_k^0]$, $m \leq k$.
- ϕ is a detailed selection condition, expressed as a conjunction of atoms $\phi_1 \wedge \dots \wedge \phi_k$.
- M_m^0 is a measure, $m \leq k$.
- Each of the levels L_α and L_β belongs to a dimension of $\mathbf{DS}^0$.
- $agg \in \{sum, min, max, count, avg\}$.

The semantics of a cube query in terms of SQL over a star schema are:

```
SELECT L_α, L_β, agg(M_m^0)
FROM DS^0, D^γ
WHERE φ AND φ_ω
GROUP BY L_α, L_β
```

where $\mathbf{D}^\gamma$ is the set of all the necessary dimension tables D_i that appear in the body of the query, $\mathbf{D}^\gamma \subseteq \{D_1, \dots, D_n\}$ and the expression ϕ_ω is a conjunction of atoms of the join conditions between DS^0 and the tables of $\mathbf{D}^\gamma$ (in the extreme case, this involves all dimension tables and $\mathbf{D}^\gamma = \{D_1, \dots, D_n\}$).

The respective semantics in terms of relational algebra are:

$$c = \gamma_{L_\alpha, L_\beta}^{agg(M_m^0)}(\sigma_\phi(DS^0 \bowtie \mathbf{D}^\gamma))$$

where $\gamma_{\mathbf{A}}^{agg(M_i)}$ denotes the grouping operator where the aggregate function agg is applied to a measure M_i over a set of grouper attributes $\mathbf{A}$.

For clarity, we summarize here the assumptions we have made for the query class of the supported cube queries:

- We work with cube queries that involve a single measure.
- We assume strictly two aggregator levels for the result; this allows a straightforward tabular representation of the result in a 2D screen.
- We assume that the selection condition is defined as the conjunction of a set of atomic formulae, *one per dimension*, each of which is of the form $L = v$, with L being a dimension level and v being a valid value for this level.

Naturally, the above list implies that extending the query class to multiple measures and groupers as well as to selection conditions involving disjunctions or level comparisons is a research challenge for future work.

A note on data presentation is due at this point. Although there are several methods that we can employ to visualize results, like for example scatter plots on a 2D space or bar charts with multiple data series, we would like to stress once again that any such visualization methods *are complementary* to the actual data [10]. So, in the rest of our deliberations, we treat the results of a cube query of the above form as being visualized in tabular format with the values of L_α as rows and the values of L_β as columns. Expanding the method for more than two dimensions (via the typical nesting of dimensions in rows and columns) is part of future work.

3.2. *Act I: putting things in context – or "How good is the original cube compared to its siblings?"*

In this subsection, we deal with the first of the acts. The main purpose of the first act is to provide a context for the original query. So, we compare the marginal aggregate results of the original query to the results of "sibling" queries that use "similar" values in their selection conditions (to be explained right next).

Method: We assume an original query and we want to compare its results with similar queries. We define a sibling query as a query with a single difference to the original: instead of an atomic selection formula $L_i = v_i$, the sibling query contains a formula of the form $L_i \in children(parent(v_i))$.

Formally, given an original query

$$q = (\mathbf{DS}^0, \phi_1 \wedge \ldots \phi_\alpha \wedge \ldots \wedge \phi_k, [L_\alpha, L_\beta], agg(M)),$$
$$\phi_i : L_i = v_i, \quad i = 1, \ldots, k$$

a new query q^s is a sibling query if it is of the form

$$q^s = (\mathbf{DS}^0, \phi_1 \wedge \ldots \phi_\alpha^* \wedge \ldots \wedge \phi_k, [L_\alpha, L_\beta], agg(M)),$$
$$\phi_i : L_i = v_i, \quad i = 1, \ldots, x-1, x+1, \ldots, k, \phi_\alpha^* : L_{\alpha+1} = anc_{L_\alpha}^{L_{\alpha+1}}(v)$$

Naturally, if q originally has k atomic selections, it also has k sibling queries.

To compare the results of the original query to the ones of its siblings, one would need to lay out all the k sibling queries on the same screen and visually inspect their differences. This becomes too hard to exploit as k increases – in fact, even with a very small k (e.g., $k = 2$) it can be too hard to be able to visually compare the results. So we need to resort to auxiliary comparisons that provide the context needed. To this end, we introduce two *marginal sibling queries*, one for each aggregator. Each time, we keep one of the two aggregators, and the other becomes L_α. If we combine this with the fact that the new selection condition ϕ_α^* restricts L_α to the siblings of the original value v, then the resulting 2D matrix has one of the original aggregators in one of its two dimensions and the siblings of v on the other. This way, the marginal values of the original query on one of the two aggregators are compared to the respective marginal values of the siblings.

Formally, given an original query

$$q = (\mathbf{DS}^0, \phi_1 \wedge \ldots \phi_\alpha \wedge \ldots \wedge \phi_k, [L_\alpha, L_\beta], agg(M)),$$
$$\phi_i : L_i = v_i, \quad i = 1, \ldots, k$$

its two *marginal sibling queries* are

$$q_\alpha^s = (\mathbf{DS}^0, \phi_1 \wedge \ldots \phi_\alpha^* \wedge \ldots \wedge \phi_k, [L_\alpha, L_\alpha], agg(M)),$$
$$\phi_i : L_i = v_i, \quad i = 1, \ldots, x-1, x+1, \ldots, k, \phi_\alpha^* : L_{\alpha+1} = anc_{L_\alpha}^{L_{\alpha+1}}(v)$$

$$q_\beta^s = (\mathbf{DS}^0, \phi_1 \wedge \ldots \phi_\alpha^* \wedge \ldots \wedge \phi_k, [L_\alpha, L_\beta], agg(M)),$$
$$\phi_i : L_i = v_i, \quad i = 1, \ldots, x-1, x+1, \ldots, k, \phi_\alpha^* : L_{\alpha+1} = anc_{L_\alpha}^{L_{\alpha+1}}(v)$$

Example. The original query is expressed as

$$q = (\mathbf{DS}^0, W.L_2 = \text{'With-Pay'} \wedge E.L_3 = \text{'Post-Sec'}, [W.L_1, E.L_2], avg(Hrs))$$

In the reference example, slides ❷ and ❸ involve the two marginal subqueries – see for example the former with the selection set to the *parent*('With-Pay') (i.e., setting the workclass dimension one level higher than the original query, to $E.L_4$) and the grouper pertaining to workclass to the level of 'With-Pay' (i.e., $E.L_3$):

$$q^1 = (\mathbf{DS}^0, W.L_2 = \text{'With-Pay'} \wedge E.L_4 = \text{'ALL'}, [W.L_1, E.L_3], avg(Hrs))$$

The equivalent SQL statements are:

Original q query	Query q^1
`SELECT w.L1, e.L2, AVG(a.hours_per_week)`	`SELECT w.L1, e.L3, AVG(a.hours_per_week)`
`FROM adult a, education e, work_class w`	`FROM adult a, education e, work_class w`
`WHERE w.L2='With-Pay' AND e.L3 = 'Post-sec'`	`WHERE w.L2='With-Pay' AND e.L4='ALL' AND`
`a.education=e.L0 AND a.work_class=w.L0`	`a.education=e.L0 AND a.work_class=w.L0`
`GROUP BY w.L1, e.L2;`	`GROUP BY w.L1, e.L3;`

Original q query

```
SELECT w.L1, e.L2, AVG(a.hours_per_week)
FROM adult a, education e, work_class w
WHERE w.L2='With-Pay' AND e.L3='Post-sec' AND
  a.education=e.L0 AND a.work_class=w.L0
GROUP BY w.L1, e.L2;
```

Query q^{gov}

```
SELECT w.L0, e.L2, AVG(a.hours_per_week)
FROM adult a, education e, work_class w
WHERE w.L1='Gov' AND e.L3='Post-Sec' AND
  a.education=e.L0 AND a.work_class=w.L0
GROUP BY w.L0, e.L2;
```

Query q^{prv}

```
SELECT w.L0, e.L2, AVG(a.hours_per_week)
FROM adult a, education e, work_class w
WHERE w.L1='Prv' AND e.L3='Post-Sec' AND
  a.education=e.L0 AND a.work_class=w.L0
GROUP BY w.L0, e.L2;
```

Query q^{s-e}

```
SELECT w.L0, e.L2, AVG(a.hours_per_week)
FROM adult a, education e, work_class w
WHERE w.L1='s-e' AND e.L3='Post-Sec' AND
  a.education=e.L0 AND a.work_class=w.L0
GROUP BY w.L0, e.L2;
```

Fig. 5. Generation of queries for Act II (in SQL).

3.3. *Act II: explaining variation – or 'drilling into the breakdown of the original result'*

The purpose of Act II is to help the user understand why the situation is as observed in the original query. In order to shed some more light to what is happening, we drill in the details of the cells of the original result in order to inspect the internals of the aggregated measures of the original query.

Assume a cube query

$$q = (\mathbf{DS}^0, \phi_1 \wedge \ldots \wedge \phi_k, [L_\alpha, L_\beta], agg(M)), \quad \phi_i: L_i = v_i, \ i = 1, \ldots, k$$

and its result, visualized as a 2D matrix. Then, each cell c of this result is characterized by the following cube query:

$$q^c = (\mathbf{DS}^0, \phi_1 \wedge \ldots \wedge \phi_k \wedge \phi_c, [L_\alpha, L_\beta], agg(M)),$$
$$\phi_i: L_i = v_i, \ i = 1, \ldots, k,$$
$$\phi_c: \phi^c_\alpha \wedge \phi^c_\beta = (L_\alpha = v^c_\alpha \wedge L_\beta = v^c_\beta)$$

For each of the aggregator dimensions, we can generate a set of *explanatory drill in queries*, one per value in the original result:

$$q^{\alpha_i} = (\mathbf{DS}^0, \phi_1 \wedge \ldots \wedge \phi_k \wedge \phi^{\alpha_i}, [L_{\alpha-1}, L_\beta], agg(M)),$$
$$q^{\beta_i} = (\mathbf{DS}^0, \phi_1 \wedge \ldots \wedge \phi_k \wedge \phi^{\beta_i}, [L_\alpha, L_{\beta-1}], agg(M)),$$

with α_i and β_i being all the values of the original result for the grouper levels. So, each of the two slides has a set of such queries.

Example. Observe slide ⓔ where we drill-down for values Gov, Private and Self-emp via the explanatory drill in queries for dimension Work.

$$q^{gov} = (\mathbf{DS}^0, W \cdot L_2 = \text{'With} - \text{Pay'} \wedge W \cdot L_1 = \text{'Gov'} \wedge E \cdot L_3 = \text{'Post} - \text{Sec'}, [W \cdot L_0, E \cdot L_2], avg(Hrs))$$

$$q^{prv} = (\mathbf{DS}^0, W.L_2 = \text{'With} - \text{Pay'} \wedge W.L_1 = \text{'Private'} \wedge E.L_3 = \text{'Post} - \text{Sec'}, [W.L_0, E.L_2], avg(Hrs))$$

$$q^{s-e} = (\mathbf{DS}^0, W \cdot L_2 = \text{'With} - \text{Pay'} \wedge W \cdot L_1 = \text{'s} - \text{e'} \wedge E \cdot L_3 = \text{'Post} - \text{Sec'}, [W \cdot L_0, E \cdot L_2], avg(Hrs))$$

Observe that due to the fact that this is the special case where selection conditions involve grouper values at finer levels of detail, we have completely removed the atomic formula of the dimension that we drill-down ($W \cdot L_2$ = 'With – Pay'). The equivalent SQL statements are depicted in Fig. 5.

3.4. *Highlights and text*

As already mentioned, the extraction of highlights is orthogonal to the query that creates the results of a slide. Once the results of the query are computed and organized in a 2D matrix, we utilize a palette of highlight extraction methods that take a 2D matrix as input and produce important findings as output. This way, (a) we can reuse highlight extraction methods to all the query results, independently of the Act or the query that has been executed, and, (b) we can gracefully extend the palette of highlight extraction methods with more results. We have implemented a small number of highlight extraction methods for the moment that include the highlighting of the top and bottom quartile of values in a matrix, the absence of values from a row or column, the domination of a quartile by a row or a column (i.e., when all the values of a quartile appear in a certain row or column), the identification of min and max values, etc. Clearly, there is a vast area of enriching this palette (trend analysis, correlations, relative relationships of rows and columns, to name just a few); however, implementing the full spectrum of such techniques can be done with diligence as part of future work.

Text is constructed by a Text Manager that customizes the text per Act, by plugging values to a template that comes with each act. Compare the following excerpt with the text of slide ⓔ in Fig. 1.

In this slide, we drill-down one level for all values of dimension <dim> *at level* <l>. *For each cell we show both the* <agg> *of* <measure> *and the number of tuples that correspond to it ...*

3.5. *Creation of CineCubes*

Having explained all the individual steps, we now move on to discuss the overall process for creating a CineCube movie. In its current configuration, a CineCube movie

```
Algorithm Construct Operational Act
Input: the original query over the appropriate database
Output: a set of an Act's episodes fully computed

  1. Create the necessary objects (act, episodes, tasks, subtasks) appropriately
     linked to each other
  2. Construct the necessary queries for all the subtasks of the Act, execute
     them, and organize the result as a set of aggregated cells (each including its
     coordinates, its measure and the number of its generating detailed tuples)
  3. For each episode
      (a) Calculate the visual presentation of cells
      (b) Calculate the cells' highlights
      (c) Produce the text based on the highlights
      (d) Produce the audio based on the text
```

Fig. 6. Constructing an Operational Act.

includes three kinds of acts: the *Introductory Act* (including the introductory slide), three *Operational Acts* including the act involving the original query and the two acts for the management of complementary queries, and, finally, a *Summary Act* with a summary slide with all the important highlights of the previous three acts.

Overall the method includes the following steps:

1. Construct Introductory Act
2. For all the Operational Acts, execute the *Construct Operational Act* algorithm that calculates the Act's contents (result visualization, highlights, text and audio)
3. Construct Summary Act in the end
4. Wrap-up the Acts in a PowerPoint movie

The first step of the method, the construction of the *Introductory Act*, is trivial. The second step, the computation of the contents and presentation of the *Operational Acts*, is outlined in the Algorithm of Fig. 6. The third step, which involves the construction of the *Summary Act*, is simply a slide with the text of the highlights copied to it, organized per act. Finally, the fourth step, wrapping-up the individual acts in a single report, introduces a few programmatic tasks worth mentioning here. In a nutshell, for every episode, we create a slide, with its title and contents (i.e., the 2D tables or the text, depending on the type of slide). This can be done straightforwardly with the programming facilities provided by the Apache POI. Unfortunately, though, POI does not support the management of notes, where we actually store the text of each slide, and audio. This is done by exploiting the open nature of pptx documents (see Section 4.1).

4. Software architecture and the pledge to extensibility

In this section, we start with a description of the employed technologies that allow the programmatic construction of CineCube movies; we believe this description can help the interested reader to probe further in the employed API's that come with a quite low technological barrier for the new-comer. Second, in this section, we describe the software architecture of CineCubes with the main goal of highlighting the different ways in which CineCubes is extensible. Extensibility is a key feature that should not be underestimated by no means, as it can allow a system to expand in diverse ways. In fact, in this case, the extensibility-oriented architecture of CineCubes is the backbone for all the diverse ways in which future research can be pursued. Readers who are not interested in the software-related aspects of our method may prefer to jump directly to Section 5 for the experimental assessment of our method; this can be done without a gap in the understandability of the text.

4.1. Employed technologies

One of the major goals of this paper is to highlight how we can automatically construct a CineCube presentation that includes result visualization, text and audio. So, before elaborating further in the software architecture of CineCubes, it is worth discussing the programmatic simplicity[3] of our method. In this subsection, we explain the main technologies via which our PowerPoint presentations are programmatically constructed.

Apache POI [3] is a Java API that provides several libraries to create and modify Microsoft Word, PowerPoint and Excel files. MS Office files obey the Office Open XML standards (OOXML) and Microsoft's OLE 2 Compound Document format (OLE2). More specifically, XSLF is the Java implementation of the PowerPoint 2007 OOXML (.pptx) file format in POI.

The automatic manipulation of .pptx files is relatively simple for simple tasks. See the following excerpt for creating a file and a slide:

```
XMLSlideShow ss=new XMLSlideShow();
XSLFSlideMaster sm=ss.getSlideMasters()[0];
XSLFSlide sl=ss.createSlide
  (sm.getLayout(SlideLayout.TITLE_AND_CONTENT));
XSLFTable t=sl.createTable();
t.addRow().addCell().setText(``added a cell''); ...
```

---

[3] We would insist that simplicity is a strong feature of any method, making it scalable, extensible and maintainable and fundamentally different to naiveness or superficiality.

Also, we automate the construction of text that characterizes each slide. We add the text for each slide that we create as a slide's note. At the same time, the existence of text can help us create a narrative as audio. We use the API provided by MARY [4], which is an open-source, multilingual Text-to-Speech Synthesis (TTS) platform written in Java and allows to generate one audio file per slide, simply by providing the notes of the slide as input to a method call.

```
MaryInterface m=new LocalMaryInterface();
m.setVoice(''cmu-slt-hsmm'');
AudioInputStream audio=m.generateAudio(''Hello'');
AudioSystem.write(audio, AudioFileFormat.Type.WAVE,
  new
  File(''myWav.wav'')); ...
```

It is worth mentioning here, that unfortunately, the current version of POI does not support the management of notes, where we actually store the text of each slide, and audio. Fortunately, the open nature of MS Office allows us to exploit its internal structure. It is noteworthy that each MS Office file is actually a zipped folder with a rigid structure, within which, XML and media files are located in a principled fashion. So, to deliver a presentation in the form that we wish to have it, we proceed as follows (via the appropriate *WrapUp Manager*): (i) we unzip the pptx in a temporary folder, (ii) create appropriate files for the notes in the ppt/notes/ folder, along with the necessary links that link them to their slide, (iii) do the same for audio at the ppt/media folder, and (iv) zip the folder again to a .pptx file.

Overall, despite the existence of several nuts and bolts that need fine tuning, *the main lesson learned here is that the packaging of the results of our method, one by one as slides in a presentation is attainable with neat programming facilities, already available in the Web.*

4.2. Architecture under the prism of extensibility

As already mentioned, CineCubes is built with inherent points of extensibility in its internal structure. In this subsection, we review the core parts of the internal architecture of Cinecubes and discuss how they can be extended. In Fig. 7 we depict the core classes of the tool and in Fig. 8 we depict the sequence diagram of our method, as described in Section 3.5.

Tasks and subtasks are the main mechanism via which queries are generated. The package `TaskMgr` contains the necessary classes which help us to create a new kind of Act. Here we have a *TaskMgr* class to manage the tasks. *A first point of extensibility of our method involves the creation of tasks*: the Task class is abstract to facilitate the creation of a different type of task for each new kind of *Act* via the appropriate materialization – e.g., in our implementation we have created two subclasses for *Act I* and *Act II* and one subclass to implement the original request. Another point of extensibility, with an eye to future work, is located in the abstract class *ExtractionMethod* which is used in order to provide us with results. Currently, we materialize this class via *SqlQuery* to get the result from a relational database; however, in future we can materialize it differently to get data from different sources e.g., XML or *SPARQL* files. Class *Result* keeps the query result in a 2D matrix and implements a set of functions to manipulate this matrix (Fig. 9).

One of the strongest points of extensibility and key to the operation of *CineCubes* is the enrichment of each slide with the possibility of extracting highlights from the data that it hosts. The package *HighlightMgr* hosts a dedicated *Highlight Manager* class to extract Highlights and an abstract class *Highlight*, which is open to extension for finding highlight in episodes. In our current implementation, we have created the subclasses of Fig. 10, with the obvious semantics, all of which implement a method *execute*() that takes a 2D matrix of values as input and creates lists of values where the findings are stored.

The parts of a story are glued together via the *Story* class and the respective *StoryMgr*. In the package `StoryMgr`, we host the main classes needed to create a Story. This package has the class *StoryMgr* to manage the story and the *Story* class. Also, it has the classes which implement the acts, the episodes of each act and the visualization of an episode.

There are several points of extensibility here:

- The implementation of the episodes of *Act* is performed via an abstract class *Episode*, which in our approach is materialized as *PptxSlide*, although in the future one can materialize it in other ways too, to create different types of episodes (e.g., frames in *.wmv file*).
- In a similar fashion, the *Story* class relates to the abstract class *FinalResult*, currently materialized as *PptxSlideshow* in our method and also open to extensions (such as *.wmv file*), in the future.
- The *Visual* class is an abstract class, currently materialized by the *Tabular* class, which visualizes the result as a pivot table; again, in the future we can create new kinds of visualization (such as a chart of all kinds).

Episodes are key for managing results: observe how the *Episode* class is associated with the *Highlight*, the *Audio*, the *Visual* class and the *Subtask* class (of package *TaskMrg*, not depicted here) and glues all these together.

A final point of extensibility involves the format of the result presented to the user. The package `WrapUpMgr` serves the purpose of packaging all the different parts computed by the previous packages in a single, final result delivered to the user. Again, this package is constructed in an extensible way as it contains the abstract class *WrapUpMgr* which, then, has to be materialized by a subclass in order to construct the proper format for a story. Currently, we create the subclass *PPTXWrapUpMgr* which returns to the user a Microsoft PowerPoint presentation.

4.3. Extending the set of result formats

One possible way to extend the tool involves adding more kinds of formats for the final result (e.g., as documents, movies, etc.). To achieve this extension, one should extend the following abstract classes by materializing the appropriate subclasses dedicated to the new format:

- the abstract class *Episode* should be appropriately materialized in order to shape the results into presentable

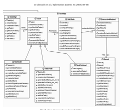


Fig. 9. Class diagram for package TaskMgr.

4.6. *Assessing the Extensibility of our framework*

In Fig. 12, we present the programming effort which was needed in order to extend the current approach of our method for adding Act II. As already mentioned, adding a new kind of Act requires to create **one** new class and to implement **three** methods. Moreover, in order to create a new kind of highlight, we must create **one** new class and implement **two** methods. We believe that in summary, the programming effort to extend our method in each flavor of extensibility is too low.

5. Experiments

In this section, we describe our experimental assessment of the CineCubes tool.

5.1. *Experimental setup*

For all our experiments, we have experimented with the Adult (a.k.a census income) data set referring to data from 1994 USA census. The data set in its cleansed version (after uncertain and NULL values are removed) comprises 30 162 tuples of the 1994 USA census. There are 8 dimensions (*Age*, *NativeCountry*, *Education*, *Occupation*, *Maritalstatus*, *Workclass*, *Gender*, and *Race*) in the data set and a single measure, *HoursperWeek*. The hierarchies for the dimensions *Education* and *Work class* are depicted in Fig. 2. The hierarchies for the dimensions *Occupation*, *Marital status*, *Gender*, and *Race* are depicted in Fig. 13 and the hierarchy of dimension *Native Country*, except the level 0 (which includes too many values), is depicted in Fig. 14. The dimension *Age* is organized in years, 5-year intervals, 10-years intervals, 20-year intervals and *.

In terms of efficiency, what we are interested to discover is where we spend more time during the generation of a CineCubes movie. As the generation of the pptx file advanced, we have carefully monitored all the individual steps of the method. Therefore, we are able to discuss the time costs of the method from two points of view: (i) concerning the individual parts of the method (results / highlight / text and audio generation etc.) and (ii) concerning the different acts of the method. To provide a thorough evaluation, we have worked with a variant number of two, three, four or five atomic selection conditions in the WHERE clause of the original query. This results in an increase in the number of slides in Act I as the number of atomic selections in the WHERE clause increases: since Act I compares with sibling values of the selection values, each selection condition adds two extra slides, where the siblings of the involved value are grouped for the two groupers (Fig. 15). Instead, the slides of Act II remain in all cases constant, which is consistent to the essence of the method, which drills down the grouping levels of the original query. All experiments have taken place in a conventional PC

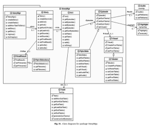


Fig. 11. Class diagram for package StoryMgr.

	# new classes	# modified classes	# new methods
new Act	1	1	2 (# new) + 1 (# modified)
new Highlight	1	1	1 (# new) + 1 (# modified)

Fig. 12. *Assessment of the extensibility effort for CineCubes.*

algorithms that try to find interesting highlights and create representative textual descriptions for them.

5.3. *Analysis of results per act*

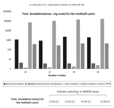
The second goal of our study was to find out how time is divided within the acts of the story. Again, we have measured the time needed to produce each Act of the story (measured in milliseconds). We depict our findings in Fig. 17.

As the number of selection conditions rises, each time we have two extra slides at Act I (the number of slides of each try is depicted in parentheses at the header of the table in Fig. 17). Clearly, we can observe that the time of each Act is increasing as the number of atomic selection conditions increases. Moreover, the construction of Act I in three of the four cases takes more time than the construction of the others – only in the case when we have two atomic selection conditions, the construction of Act II takes about 90 ms more. In addition, the time needed to create Act II is practically stable, independently of the number of atomic selection conditions in the WHERE clause.

In Fig. 17, observe that as the number of slides increases (2 extra slides each time) Act I increases with significant rate; the Summary Act behaves similarly, albeit with a lower increase. Both these effects are mainly due to the text and audio generation (as already mentioned). The linearity of the increase for Act I can safely be attributed to the cost of the extra slides that are added each time to the Act (in fact, this is also corroborated by the detailed measurements per individual slide that are not included here).

Also in Fig. 17, we can observe that the Summary Act needed more time than the Act II in three of the four cases. This happens because the Summary Act, as described in Chapter 2, has all the highlights of the story (i.e., all the text for these highlights) which must be also converted to sound. Once again the text to speech API dominates the time of our result.

Why does the Summary Act increase with a smaller rate than Act I? This happens because the Summary Act has only the highlights of all episodes (and not the entire text). Due to the text that dresses and contextualizes the highlights in each slide, Act I has always more words to be



Time breakdown (msec) for the method's parts	# atomic selections in WHERE clause			
	2 (10 sl.)	3 (12 sl.)	4 (14 sl.)	5 (16 sl.)
Result Generation	1169.00	881.40	2263.91	1963.68
Highlight Extraction & Visualization	4.41	3.60	3.67	3.74
Text Creation	1.32	1.42	1.80	2.35
Audio Creation	71463.21	104634.27	145004.20	169208.59
Put in PPTX	378.24	285.89	452.74	460.55

Fig. 16. Time breakdown (ms) for the method's parts.

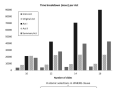
them in SQL. The users that participated in the study were 12 PhD students from our Department, all of which were experienced in data management and statistics.

The user study consisted of four phases.

Phase 0 (*warm-up*): In the first phase, the users were familiarized with the data set and the tools. To this end, we presented the data set, its dimensions and levels. We also gave a demo of how to pose queries to the systems. We explained to the users that they could use any combination of (a) slideshow, (b) browsing through the slides, and (c) reading a printout of a query result or a CineCubes report. All users were given a pamphlet reminding the basics of the above. Then, the users returned to their offices, where they all had ample networking, computing and printing facilities to work with the next of the phases.

Phase 1 (*simple OLAP functionality*): The first part of the evaluation was to ask the users to prepare a report on a specified topic. The report should contain (a) a bullet list of key, highlight findings, (b) a text presenting the overall situation and, (c) optionally, any supporting statistical charts and figures to elucidate the case better.

Phase 2 (*CineCubes functionality*): We assigned the same task to the users, but now, they had CineCubes available. Both the simple querying system and CineCubes were at the disposal of the users in order to pose auxiliary requests for simple queries or CineCubes reports. To speed up the



Farm Investment (euro) per Ind			
Farm A	2,780.00	4,345.12	4,621.71
Farm B	7,901.27	8,823.29	8,914.67
Farm C	19,661.78	42,821.29	78,615.32
Farm D	12,251.69	23,423.34	23,623.94
Agricultural Year	19,321.23	27,921.21	42,511.23

Fig. 10. Farm investment per Ind and Agricultural Year.

Table 1
Farm investment (euro) per Ind and Agricultural Year

Farm Investment (euro) per Ind	Agricultural Year		
	19,321.23	27,921.21	42,511.23
Farm A	2,780.00	4,345.12	4,621.71
Farm B	7,901.27	8,823.29	8,914.67
Farm C	19,661.78	42,821.29	78,615.32
Farm D	12,251.69	23,423.34	23,623.94

assess, we are provided a link with a number of free, open-source utilities.

When implemented as intended, then the work-load of the Quartermaster will be reduced. Quartermaster is a questionnaire within the social network and its aim is to collect information about the Quartermaster and its supplier. Information for the 'new market' is complete. Quartermaster is a questionnaire within the social network and its aim is to collect information about the Quartermaster and its supplier. Information for the 'new market' is complete. Quartermaster is a questionnaire within the social network and its aim is to collect information about the Quartermaster and its supplier. Information for the 'new market' is complete.

Table 2
Farm investment (euro) per Ind and Agricultural Year

Farm Investment (euro) per Ind	Agricultural Year		
	19,321.23	27,921.21	42,511.23
Farm A	2,780.00	4,345.12	4,621.71
Farm B	7,901.27	8,823.29	8,914.67
Farm C	19,661.78	42,821.29	78,615.32
Farm D	12,251.69	23,423.34	23,623.94

Quartermaster is a tool of C. A. Smith. Fusing the carrot while the Quartermaster is hot is a task for Quartermaster. Quartermaster is a tool of C. A. Smith.

Test: Analysis of Covariance plot

In the part of the questionnaire, he sees sure, what he prefers, a succession of the questions at the point of

average higher than 3. The most popular feature was Act II, with the detailed, drill-down analysis of the groupers. The users attributed this to the fact that it provided them with information they thought interesting to include in the report, as it enlarged the picture of the situation that was presented to them. The second most popular feature was the treatment of the original query (that includes coloring, and highlight extraction compared to the simple query results given to them by the simple querying system).

It has been quite interesting that the two less appreciated parts were Act I (which contextualizes the result by comparing it to similar values) and the summary act (presenting all the highlights in a single slide). Although this originally came as a surprise to us, with the benefit of the hindsight we reckon that it should not have been surprising in the first place: users love concise reporting of facts and dislike information provided in large volumes to them. The free-form comments of the users and a post-mortem discussion with them confirmed this observation. The contextualization and the summary acts provide too much information (and in fact, too many highlights). So, although the peak is in the median value (3), the average value for both Act I and the Summary Act was 3.4 stars and the distribution of values towards the high end, the phenomenon was not so heavy tailed on the higher values as for Act II and Act 0. *Lesson learned: above all, be concise!*

The textual part was quite appreciated by most of the users; at the same time, out of 5 users that worked with audio, the result was split in half in terms of likes and dislikes. This is both due to the quality of the produced audio by the TTS and the quality of the text that is served to it as input. A lesson learned here is that *audio seems to be useful for some users but not for all*; so, it should be optional, which can provide gains in terms of efficiency without affecting effectiveness.

5.4.3. *Evaluation of the produced reports*

The users were also asked to assess the quality of the produced report with the benefit of the hindsight. The results are depicted in Fig. 20.

Overall, the distribution appears shifted by one star upwards, with the median shifting from 3 to 4. The average value was raised from 3 to 3.7 which is a 23% improvement of quality. The free-form comments indicated that the score would have been higher if the tool automatically produced graphs and charts (an issue of small research but high practical value).

5.4.4. *Time considerations*

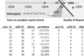
We asked the users to measure the time they spent for the creation of each report. In Fig. 21 we depict the actual data as well as their visual representation. The graphical representation of Fig. 21 compares the benefit in time (*x*-axis) over the benefit in stars (*y*-axis). In other words, does it pay off to spend more time working with the system for the quality of the report one gets? The diagonal line splits the plane in two parts: the right, green part is the area where you get more quality for the time you invest; the left, rose part is an area of loss. The intensely colored parts of the two areas are parts with two-fold benefit (more quality for less time) or loss (less quality for more time).

The findings are quite interesting. A first very interesting observation lies in the fact that CineCubes did not result in clear time gains, as we would expect. In fact, there was a

Fig. 18. Difference in words and execution time for Act I and Summary Act.

Fig. 19. Evaluation of the usefulness of CineCubes' parts, in a scale of 1 (worst) to 5 (best); *x*-axis depicts each score and *y*-axis the number of users that assigned it.

Fig. 20. Evaluation of the original and the Cinecubes' report, in a scale of 1 (worst) to 5 (best); x-axis depicts each score and y-axis the number of users that assigned it.



User id	Time to complete report (mins) w/o CC	with CC	Δtime	pct Δtime	Quality of Report (stars) w/o CC	with CC	ΔStars	pct Δstars
3	10	20	-10	-100,00%	3	3	0	0,00%
4	11	45	-34	-309,09%	2	3	1	50,00%
2	23	45	-22	-95,65%	2	4	2	100,00%
5	23	25	-2	-8,70%	3	4	1	33,33%
21	25	45	-20	-80,00%	3	4	1	33,33%
8	25	25	0	0,00%	3	4	1	33,33%
12	30	26	4	13,33%	4	4	0	0,00%
17	60	40	20	33,33%	2	5	3	150,00%
6	70	40	30	42,86%	4	4	0	0,00%
1	71	25	46	64,79%	3	3	0	0,00%
15	100	45	55	55,00%	3	3	0	0,00%
16	105	35	70	66,67%	4	3	-1	-25,00%

Fig. 21. Evaluation of the time gains versus the quality gains for the construction of the report with and without CineCubes. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

large number of people who spent more time with CineCubes than with the simple querying system! Although this originally did strike us as a failure, a better look at the data (and the graph) refutes this result. When we sorted the data by time spent without CineCubes (second column), it was clear that the users who demonstrated this kind of time loss were the ones who spent too little time (way less than the rest) for their original report. The small amount of time devoted to the original report, skyrockets the percentage deficit (a user who spends 10 min for the original report and 20 min for Cinecubes, gets a 100% time penalty). At the same time, this resulted also in an original report of rather poor quality, and significant improvements in the quality of the report, too. This also explains why there are no users with dual loss. Again, the explanation for the time increase is that the users spent extra time to go through the highlights offered by CineCubes.

A second observation concerns the people who spent less time with CineCubes than without it. These are people who invested more time working with data than the previous group. In all but one cases, there was no loss of quality for this group of users. So, clearly, for the people who would spend at least 30 min for their original report, there is a benefit in time gains. In fact, in all but one cases, the benefit rises with the time spent in the original report (the relationship between $\Delta Time$ and the $pct\Delta Time$ for the people with a positive time gain is almost linear, with a Pearson correlation of 0.940; the same applies for the correlation of the time spent without Cinecubes and $pct\Delta Time$ with a Pearson correlation of 0.868). It is interesting that because these

Bégué et al. (p. 11) work along the same original and provide some very interesting insights for big-input-little-output systems. Their paper is also one of the few that use the new widely-adopted R, the paper gives a few examples of how to use the language for remote sensing analysis (specifically, have a look at the *shotGun* method in the *RStoolbox* package). Finally, they also provide a number of interesting rules of thumb (see e.g. p. 17) for the application of machine learning algorithms and a detailed survey of relatively recently published studies that have used RF and SVM algorithms (see p. 18 and 19) that could be useful for the creation of a review RSE article.

3.2. Advanced RS/GIS products

Two papers (Bégué et al., p. 11; Lambert et al., p. 1) explore the potential for the creation of remote sensing products that are tailored to the needs of smallholder agriculture. Lambert et al. (p. 1) also explore the development and uptake of technology in a series of villages in Mali, West Africa (see Section 3.3).

Technical limitations like MAUP (Lambert et al., p. 1) and the need to ground-truth (Bégué et al., p. 11) are also highlighted. The reasons for these or successes selected are also explored, but in a way that is not always comparable, with a focus more on differences in outputs than approaches.

Bégué et al. (p. 11) also provide a useful review of the literature which, to the best of my knowledge, is the most comprehensive review of RS and smallholder agriculture to date. The review considers three main questions: (1) What are the main challenges for RS in smallholder agriculture? (2) What are the current RS approaches to overcome these challenges? (3) What are the gaps in knowledge and the future research needs? The review is well-structured and provides a good overview of the literature. Some of the key findings are that: (1) there is a lack of ground-truth data; (2) there is a lack of standardisation in methods; (3) there is a lack of operational products; and (4) there is a lack of uptake of products by farmers. The review also highlights the importance of considering the socio-economic context of RS applications, and the need for a more holistic approach to the development of RS products.

3.3. Tool uptake and end-users

Ryan et al. (p. 1) present an approach to participatory mapping of agricultural systems in the Lake Chad Basin, Africa. The authors employ a digital sketch-map approach to collect data on land use and land cover, farming practices, ecology and socio-demographic and taboo for villages. The approach is well-described and could be useful for other projects around. The method relies on paper, an object which is not always available in the field, and the digitisation of the paper maps is a bit crude. This is another area where R could be useful, for example, to create custom polygons in different layer-sheets.

3.4. Data curation, metadata standardisation and cloud computing

Lambert et al. (p. 1) present a novel way to collect data in an approach termed data curation. Whereas the words curation and curator are usually used in the context of museums, the term is used to describe the work of people who curate the data in such a way that it is easily accessible, and more fundamentally, ‘appropriate’ (visualisation, flow to end-users, interpretation, etc.) for the intended purpose.

Two papers (Bégué et al., p. 11; Lambert et al., p. 1) highlight the use of cloud computing platforms, such as Google Earth Engine (GEE) and Open Data Cube (ODC). Bégué et al. (p. 11) also highlight the use of open source software packages, such as the Sentinel Application Platform (SNAP) and the Orfeo Toolbox (OTB). The use of open source software packages is also highlighted by Ryan et al. (p. 1) who use QGIS for their mapping exercise. The use of open source software is very important, and I think that we should encourage its use where possible.

Lambert et al. (p. 1) also highlight the importance of metadata standards, and the use of the Open Data Kit (ODK) software suite to successfully standardise on a entry for each record. Here, I think that we could do more to encourage the use of these tools, and also, use and take into consideration, data-driven approaches to metadata standardisation. Bégué et al. (p. 11) also highlight the importance of metadata standards, and the need for a more holistic approach to the development of RS products.

3.5. Other

In a highly cited paper, Lambert et al. (p. 1) provide a useful review of the literature on the adoption of technology in agriculture, and in particular, the challenges that smallholder farmers face in adopting new technologies. The authors also provide a succinct summary of the key factors that influence the adoption of technology in crisis and conflict contexts. For RS, despite one of the most commonly-cited reasons for the non-adoption of RS products being the cost, the cost of RS products is not always the main barrier to adoption. Other factors, such as the lack of skills and knowledge, the timeliness of the data, the ease of use, and the relevance of the data, are also important factors that influence the adoption of RS products. The authors also provide a useful framework for the implementation of RS projects that can be used to guide the development of RS products. The framework is based on the idea that the adoption of RS products is a process that can be broken down into a series of steps, and that each step has its own challenges and opportunities. The framework is also flexible, and can be adapted to the specific needs of the end-users.

3.6. Tool uptake (end-user) cont.

Ryan et al. (p. 1) present an approach to participatory mapping of agricultural systems in the Lake Chad Basin, Africa. The authors employ a digital sketch-map approach to collect data on land use and land cover, farming practices, ecology and socio-demographic and taboo for villages. The approach is well-described and could be useful for other projects around. The method relies on paper, an object which is not always available in the field, and the digitisation of the paper maps is a bit crude. This is another area where R could be useful, for example, to create custom polygons in different layer-sheets.

3.4. Data curation, metadata standardisation and cloud computing

Lambert et al. (p. 1) present a novel way to collect data in an approach termed data curation. Whereas the words curation and curator are usually used in the context of museums, the term is used to describe the work of people who curate the data in such a way that it is easily accessible, and more fundamentally, ‘appropriate’ (visualisation, flow to end-users, interpretation, etc.) for the intended purpose.

Two papers (Bégué et al., p. 11; Lambert et al., p. 1) highlight the use of cloud computing platforms, such as Google Earth Engine (GEE) and Open Data Cube (ODC). Bégué et al. (p. 11) also highlight the use of open source software packages, such as the Sentinel Application Platform (SNAP) and the Orfeo Toolbox (OTB). The use of open source software packages is very important, and I think that we should encourage its use where possible.

Lambert et al. (p. 1) also highlight the importance of metadata standards, and the use of the Open Data Kit (ODK) software suite to successfully standardise on a entry for each record. This is another area where I think that we could do more to encourage the use of these tools, and also, use and take into consideration, data-driven approaches to metadata standardisation. Bégué et al. (p. 11) also highlight the importance of metadata standards, and the need for a more holistic approach to the development of RS products.

3.5. Other

In a highly cited paper, Lambert et al. (p. 1) provide a useful review of the literature on the adoption of technology in agriculture, and in particular, the challenges that smallholder farmers face in adopting new technologies. The authors also provide a succinct summary of the key factors that influence the adoption of technology in crisis and conflict contexts. For RS, despite one of the most commonly-cited reasons for the non-adoption of RS products being the cost, the cost of RS products is not always the main barrier to adoption. Other factors, such as the lack of skills and knowledge, the timeliness of the data, the ease of use, and the relevance of the data, are also important factors that influence the adoption of RS products. The authors also provide a useful framework for the implementation of RS projects that can be used to guide the development of RS products. The framework is based on the idea that the adoption of RS products is a process that can be broken down into a series of steps, and that each step has its own challenges and opportunities. The framework is also flexible, and can be adapted to the specific needs of the end-users.

The taxonomy organizes the characteristics of visualizations in seven groups, organized in three families. The first family-group concerns the genre of the visualization (e.g., slide show, comic strip, poster, film, etc.). The second family concerns the techniques used for the visualization of a story. The first group in this family concerns the structure of the visualization (consistent visual platforms, progress indication, etc.), the second group concerns visual highlight features (zoom, motion, audio, etc.), and the third group concerns the way transitions among visualized information is made (e.g., via continuity principles, or animation). The third family concerns the structuring and interactivity of a visual presentation. The first group in this family concerns the order of the presentation's parts, the second group concerns the modes of interactivity via which the user can interact with the visualization (e.g., hovering tooltips, tacit tutorials, navigation buttons, etc.) and the third group concerns the messaging tools employed to inform the user on important parts of the presentation (e.g., captions, annotations, summaries, etc.). One typical difference between traditional storytelling and data narration that [6] highlights concerns the potential for interactivity in the latter. In fact, one of the main problems of data narration is built around the interactivity issue: how does one balance (a) a certain amount of control that the author needs to preserve and impose in order to manage to tell the story in the end, versus (b) the need of the users to go through alternative explorations, pose verification or explanatory questions, and in any case, work with the data in ways not already present in the linear storytelling of the author. The authors highlight three ways of interacting with the users. First, the Martini Glass principle, where the presentation starts with a user-centric part for the exploration of data, continues with a strict, non-interactive sequence to convey its core message and concludes with a large number of choices for follow-up exploration by the user. The Interactive Slideshow principle follows a typical slideshow format (i.e., a sequence of author-driven "slides") but allows some degrees of interaction within each slide, allowing thus the user to interact with the data mid-narrative. The Drill-Down Story principle gives the most degrees of interaction to the user, by presenting a central theme as a portal for interactive data exploration.

In a similar trend, Hullman and Diakopoulos [23] conduct a similar study to provide a taxonomy of techniques ("rhetoric") used to illustrate or obscure information. Information access rhetoric methods concern which data to include or exclude, and at what level of aggregation or abstraction. Provenance rhetoric methods concern ways to highlight or obscure the source of information and the uncertainty involved in the reporting of facts and estimations. Mapping rhetoric concerns techniques like visual metaphors, contrast, color coding etc., used to map data to a visual representation that conveys a message. Linguist-based rhetoric methods are focused on the textual level and concern how the presenter uses text to make a point or make the user be involved.

A central role in data narration is played by interactive visual analytics that facilitate the extraction of information from data via interactive visualization and automated information mining. Ben Shneiderman gave the famous "Visual Information Seeking Mantra" back in 1996 [24], as the foundation of visual design: Overview first, zoom and filter, then details-on-demand. Fifteen years later, in [25], Heer and Schneiderman provide a taxonomy of visual tools that facilitate the interaction of a user with data, in what the authors call "analytic dialogues". The taxonomy consists of 12 typical task types grouped in categories as follows: (a) data and view specification (visualize, filter, sort, and derive), practically facilitating the setup and focus of a data visualization task, (b) view manipulation (select, navigate, coordinate, and organize), practically covering the part where a user explores the presented data interactively, and (c) analysis process and provenance (record, annotate, share, and guide), covering the part where the user has gained insight and is preparing his data-driven story's presentation.

6.5. Visualization for OLAP

Although database research has not dealt with visualization issues in depth, there exist some efforts towards visualizing data cubes and OLAP results.

Visualization for mobile devices has been a central motive for these efforts that go back to early '00s. A method reported by Maniatis et al. [11] introduced a formal presentation model for OLAP and focused on the problems coming from the small display of mobile devices (putting focus to important values). A method by Cuzzocrea et al. [26,27] addressed the lack of storage facilities in mobile devices and placed its emphasis on working with a highly summarized view of the data that is representative enough for the user to work with. Recently, the authors of [28] made another attempt to the problem of presenting data in mobile devices by addressing problems caused by the network latencies. So, the authors propose a software architecture and method for in-place calculations and data prefetching in order to accommodate the need of immediate data presentation in cases like top-k scrolling or moving average calculations.

Apart from efforts particularly tailored for mobile devices, there have been efforts for visualizing the results of hierarchical multidimensional data in the broader area of OLAP. As expected, the main obstacle to overcome is the limitation of the dimensionality of the display media (screen and paper are 2D) as opposed to the multidimensional nature of the data. The presence of hierarchies makes the task even harder. The employed techniques range from traditional ones to quite elaborate visualization schemes. The most scalable visualization scheme in terms of dimensionality is parallel coordinates: Inselberg in [29] discusses the possibilities provided by parallel coordinates to handle the problems of large dimensionalities. Of course, this comes at the price of a learning curve, as our visual system (and our Cartesian-based education) are oriented towards more traditional visualization schemes. So, at the other end of the spectrum, we see efforts like the one by Dymson and Florez [30] who employ traditional crosstabs to display data along with different shading levels to indicating the completeness of each value (as data come in a streaming way). In between these two extremes, we can find some other attempts. A notable effort by Vinnik and Mansmann in [31] provides a classification of visualization schemes and make a case for trees reflecting the hierarchical aggregation of data. Then, the paper proposes ways to instruct the users in their navigation of the lattice of cuboids via paths on such a tree along with other visualization methods for demonstrating measures (resulting e.g., in a bar-chart tree showing measures

in par with the navigation tree). More exotic visual schemes like Treemaps or Solar plots are discussed in [12]. The latest effort that we are aware of, in the area of OLAP visualization, comes by Ordonez et al. in [33]. The authors are based on the aforementioned hierarchical decomposition family of methods and provide a sophisticated method based on the layered, visual placement of cuboids on the lattice produced by all the possible aggregators of a fact table. Cuboids interesting to the user (via a query specification) are colorized differently; drilling-into them provides a visual overview of the values involved in the cuboid. Interestingly, this graphical representation is complemented by the visualization of the relationship of interesting results. Following [34], the authors perform statistical tests to highly similar groupings of the cube; when the difference of two similar cells in a measure is statistically important, this is highlighted as a significant result and the pair is linked via an edge on the screen.

6.6. Relationship of our work with the state of the art

Concerning all the above works, our method comes with an extensible architecture that is especially constructed with a mindset of being able to plug in more and more of them, both at the part where new queries can be added and in the part where new analyses can be performed over their results. Our Act II resembles the DIFF operator to a certain extent, in the sense that it tries to explain the reasons of the originally observed result. DIFF goes one step further, in providing maximal explanations by picking the most profitable rows. Although DIFF can be integrated in our tool, the emphasis so far has been in coming up with a prototype that can provide a reasonable CineCube movie; research results like DIFF can be integrated in the tool in subsequent tool extensions and revisions. The same applies for all the other advanced OLAP operators.

Concerning text synthesis, we avoid describing the result of a query row-by-row, as [22] does. On the contrary, we provide an extensible architecture where each highlight extraction method comes with a generic text to describe the detected highlights. Of course, improvements on the produced text are clearly part of future work.

Concerning the visualization aspect, as already mentioned, we take special care to depict the actual data for each slide that we generate, in our attempt to facilitate Tufte's motos for "data integrity" and "intense clarity via intense detail". For the moment, we constrain our visual representation of query results to a crosstab form. Of course, this representation can be accompanied by graphical representations of all kinds. In the future, interesting visualization challenges will arise if Cinecubes departs from its reporting nature (which is very well served by the 2D crosstabs) and allow interactivity (refer to the Discussion section). Then, visualization will need to consider interactive settings like [29–31,33].

We would also like to highlight that our method is synchronized with the key findings of [18], as (a) the main criterion that we use for suggestions in Act I is what [18] identified as the key characteristic feature of cube queries, i.e., the selection condition and (b) the key feature for explaining results in Act II is the second most characteristic feature, groupers.

6.7. Relationship to our previous work

A first version of this paper has appeared in [35]. In the present version of the paper, we extend [35] in the following ways. We provide a detailed explanation of the internal architecture and the extensibility mechanisms of CineCubes. We also provide an assessment on the effort needed to extend CineCubes. We discuss in detail the results of our experimental study, both in terms of the anatomy of the time spent in the different tasks and acts and in terms of usability, via a user study. We accompany these extensions, with explanations, an extended discussion of our motivations, and a detailed survey of the related literature that were not present in the short version of this paper.

7. Discussion

In this paper we have introduced CineCubes, a system that allows the automatic generation of a CineCube movie, over an OLAP database, with a simple user query as starting point. To produce a movie, we need several "episodes", text, and voice. To this end, we have automated (a) the process of complementing the original query with additional queries that provide contextualization (by comparing its results to the results of queries carrying similar information) and (b) the process of extracting meaningful relationships within the data, as we search for interesting patterns in the results of all these queries. Moreover, we have also automated the generation of text describing these findings and their conversion of this text to audio. Finally, we have shown that all the above can be packaged in a PowerPoint presentation, practically presenting a small data movie to the user.

From the practical point of view, as already mentioned, we have demonstrated the practical application of our method via a prototype. The integration of our method to existing tools should pose no significant challenges, due to the simplicity of the underlying assumptions (a simple, typical OLAP environment with dimensions and detailed cubes, queries involving simple filters and aggregations and a single user query as seed for the process to start). From the research perspective, we believe that our method creates new research ground, by bundling all the individual steps in a single-yet-extensible framework that allows data workers gain insights. This is – in our point of view – the core contribution of this paper. Naturally, as typically happens in science, this new research ground can be further expanded in a systematic way. In the following, we list a few important problems and opportunities.

Extensibility: Cinecubes comes with an extensible architecture that is especially constructed with a mindset of hosting more and more techniques both from existing and foreseeable research results in the areas of knowledge extraction, query recommendation, text analysis, trend prediction, and data visualization. We firmly believe that this extensibility can and should be exploited via a synergy with the research community in order to further enhance the benefits of this approach.

Efficiency: Scaling with data size and complexity, let along with user needs, in user time, is also necessary for an effort like this to succeed. Techniques like multi-query

optimization have a good chance to succeed, especially since we operate with a known workload of queries as well as under the divine simplicity of OLAP.

Can I be the director? Interactively maybe? Personalization and interactivity are two clear paths for extending the approach mentioned here. Interactivity, i.e., the possibility of allowing the user to intervene and semi-automatically guide the query generation can be served in many ways (e.g., the Martini Glass and the Interactive SlideShow of [6]). This user-driven interaction can be aided by incorporating extra knowledge into the report generation – e.g., via user profiles or user logs, like in [17] – that guide the users in their explorations around the basic results that they see in a CineCubes movie.

***Be compendious*; *if not*, *at least be concise*!** The single most important challenge that the research problem of answer-with-a-movie faces is the *identification of what to exclude*. The problem is not to add more and more recommendations or findings (at the price of time expenses): this can be done both effectively (too many algorithms to consider) and efficiently (or, at least, tolerably in terms of user time). The main problem is that it is very hard to keep the story both interesting and informative and, at the same time, automate the discovery of highlights and findings. To address this task, a clearly important topic of research involves the automatic merging, ranking and pruning of highlights.

Acknowledgments

The authors would like to thank the anonymous reviewers for constructive comments and suggestions.

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

Appendix A. Supplementary data

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.is.2014.12.006.

References

[1] C. Dove, S. Jones, Narrative visualization: sharing insights into complex data, in: Presented in Interfaces and Human Computer Interaction (IHCI 2012), 2012, available at: 〈http://openaccess.city.ac.uk/1134/〉.
[2] P. Hanrahan, Analytic database technology for a new kind of user – the data enthusiast, in: Keynote Speech at the ACM SIGMOD International Conference on Management of Data (SIGMOD 2012), 2012, available at: 〈http://www.graphics.stanford.edu/~hanrahan/talks/enthusiasts.pdf〉.
[3] The Apache Organization, see 〈https://poi.apache.org/〉, The Apache POI Project, 2013.
[4] DFKI, see 〈http://mary.dfki.de/〉, The MARY Text-to-Speech System, 2013.
[5] K. Kosara, J.D. Mackinlay, Storytelling: the next step for visualization, IEEE Comput. 46 (5) (2013) 44–50.
[6] E. Segel, J. Heer, Narrative visualization: telling stories with data, IEEE Trans. Vis. Comput. Graph. 16 (6) (2010) 1139–1148.
[7] C.S. Jensen, T.B. Pedersen, C. Thomsen, Multidimensional Databases and Data Warehousing, Synthesis Lectures on Data Management, Morgan & Claypool Publishers, Lexington, KY, USA, 2010.
[8] P. Vassiliadis, S. Skiadopoulos, Modelling and optimisation issues for multidimensional databases, in: 12th International Conference on Advanced Information Systems Engineering (CAiSE 2000), 2000, pp. 482–497.
[9] R. McKee, Story: Substance, Structure, Style and the Principles of Screenwriting, HarperCollins Pub., New York, NY, USA, 1997.
[10] E.R. Tufte, Visual Explanations: Images and Quantities, Evidence and Narrative, Graphics Press, Cheshire, Connecticut, USA, 1997.
[11] A.S. Maniatis, P. Vassiliadis, S. Skiadopoulos, Y. Vassiliou, G. Mavrogonatos, I. Michalarias, A presentation model and non-traditional visualization for OLAP, Int. J. Data Warehous. Min. 1 (1) (2005) 1–36.
[12] P. Marcel, E. Negre, A survey of query recommendation techniques for data warehouse exploration, in: Actes des 7èmes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2011), 2011, pp. 119–134.
[13] K. Stefanidis, M. Drosou, E. Pitoura, "You May Also Like" results in relational databases, in: 3rd International Workshop on Personalized Access, Profile Management, and Context Awareness: Databases (PersDB 2009), 2009.
[14] G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, N. Polyzotis, J.S.V. Varman, The querie system for personalized query recommendations, IEEE Data Eng. Bull. 34 (2) (2011) 55–60.
[15] V. Cariou, J. Cubillé, C. Derquenne, S. Goutier, F. Guisnel, H. Klajnmic, Built-in indicators to discover interesting drill paths in a cube, in: Proceedings of 10th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2008), 2008, pp. 33–44.
[16] A. Giacometti, P. Marcel, E. Negre, A. Soulet, Query recommendations for OLAP discovery driven analysis, in: DOLAP 2009, ACM 12th International Workshop on Data Warehousing and OLAP, Hong Kong, China, November 6, 2009, Proceedings, 2009, pp. 81–88.
[17] A. Giacometti, P. Marcel, E. Negre, A. Soulet, Query recommendations for OLAP discovery-driven analysis, Int. J. Data Warehous. Min. 7 (2) (2011) 1–25.
[18] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, E. Turricchia, Similarity measures for OLAP sessions, Knowl. Inf. Syst. 39 (2) (2014) 463–489, http://dx.doi.org/10.1007/s10115-013-0614-1.
[19] S. Sarawagi, Explaining differences in multidimensional aggregates, in: Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99), 1999, pp. 42–53.
[20] S. Sarawagi, User-adaptive exploration of multidimensional data, in: Proceedings of 26th International Conference on Very Large Data Bases (VLDB 2000), 2000, pp. 307–316.
[21] G. Sathe, S. Sarawagi, Intelligent rollups in multidimensional OLAP data, in: Proceedings of 27th International Conference on Very Large Data Bases (VLDB 2001), 2001, pp. 531–540.
[22] A. Simitsis, G. Koutrika, Y. Alexandrakis, Y.E. Ioannidis, Synthesizing structured text from logical database subsets, in: 11th International Conference on Extending Database Technology (EDBT 2008), 2008, pp. 428–439.
[23] J. Hullman, N. Diakopoulos, Visualization rhetoric: framing effects in narrative visualization, IEEE Trans. Vis. Comput. Graph. 17 (12) (2011) 2231–2240.
[24] B. Shneiderman, The eyes have it: a task by data type taxonomy for information visualizations, in: Proceedings of the 1996 IEEE Symposium on Visual Languages (VL'96), 1996, pp. 336–343.
[25] J. Heer, B. Shneiderman, Interactive dynamics for visual analysis, ACM Queue 10 (2) (2012) 30.
[26] A. Cuzzocrea, F. Furfaro, D. Saccà, Hand-olap: A system for delivering OLAP services on handheld devices, in: 6th International Symposium on Autonomous Decentralized Systems (ISADS 2003), 9–11 April 2003, Pisa, Italy, 2003, pp. 80–87.
[27] A. Cuzzocrea, F. Furfaro, D. Saccà, Enabling OLAP in mobile environments via intelligent data cube compression techniques, J. Intell. Inf. Syst. 33 (2) (2009) 95–143.
[28] T. Hsiao, W. Lai, S. Petkhulat, Data visualization on web-based OLAP, in: DOLAP 2011, ACM 14th International Workshop on Data Warehousing and OLAP, Glasgow, United Kingdom, October 28, 2011, Proceedings, 2011, pp. 75–82.
[29] A. Inselberg, Visualization and knowledge discovery for high dimensional data, in: UIDIS, 2001, pp. 5–24.