Space-bounded query approximation

Boris Cule, Floris Geerts, and Reuben Ndindi

University of Antwerp, Belgium
{boris.cule,floris.geerts,reuben.ndindi}@uantwerpen.be

Abstract. When dealing with large amounts of data, exact query answering is not always feasible. We propose a query approximation method that, given an upper bound on the amount of data that can be used (*i.e.,* for which query evaluation is still feasible), identifies a part C of the data D that (i) fits in the available space budget; and (ii) provides accurate query results. That is, for a given query Q, the query result $Q(C)$ is close to the exact answer $Q(D)$. In this paper, we present the theoretical framework underlying our query approximation method and provide an experimental validation of the approach.

Keywords: Big data query processing, query approximation, data reduction.

1 Introduction

Traditional query processing has primarily focused on the efficient computation of exact answers to queries. In applications with huge amounts of data, however, even simple queries that require a single scan over the entire database cannot be answered within an acceptable time bound. To accommodate for this, one can either try to leverage parallelism and distributed computation, settle for approximate query answering, rely on data-reduction techniques, or combinations thereof. In this paper we consider approximate query answering, or AQA for short.

Motivated by the need for big data analytics, recent work on AQA mainly concentrates on the efficient and accurate evaluation of simple aggregate queries. A recent proposal in this context is the BlinkDB system [1]. In a nutshell, BlinkDB addresses the following question: Given a query workload $\mathcal{Q}=\{Q_1,\ldots,Q_\ell\}$ consisting of aggregate queries Q_i, each equipped with an importance weight p_i, for $i\in\{1,\ldots,\ell\}$, a database D and given a storage capacity $\mathbb{B}$, what is the best set of samples $\mathcal{S}=\{S_1,\ldots,S_\ell\}$ of D that one should materialise such that (i) the samples fit in the available storage, i.e., $|S_1\cup\cdots\cup S_\ell|\leqslant\mathbb{B}$; and (ii) evaluating the queries on samples in $\mathcal{S}$ provides an accurate estimate of the exact query answer. In addition, the most important queries (*i.e.,* those with high weight) should be approximated more accurately than the less important queries (*i.e.,* those with low weight).

In this paper, we consider a similar setting as in BlinkDB but for *non-aggregate queries*. That is, we are interested in finding the best set C of tuples in the database D that one should store within the available storage capacity $\mathbb{B}$ such that $Q_i(C)$ is "close" to $Q_i(D)$ for any $i\in\{1,\ldots,\ell\}$. For this purpose, we equip databases with distance functions to measure the closeness between two databases (Section 2), replace the samples used in BlinkDB by so-called coverings C of the data, *i.e.,* sets of tuples that are within

a certain distance from the original database, and introduce the valid selection covering problem (Section 3). We then show how to estimate the size of coverings (Section 4) and how the distance between coverings and the original data propagates through the queries in the workload (Section 5). Finally, similar to BlinkDB, we identify the desired coverings of the data by means of a mixed integer linear program (Section 6). An experimental validation of our AQA framework (Section 7) concludes the paper.

2 Preliminaries

▷ *Databases.* Let $\mathcal{R}=(R_1,\ldots,R_n)$ be a relational schema consisting of n relations R_i, each having a fixed arity m_i. Let $R(A_1,\ldots,A_k)$ be a relation in $\mathcal{R}$. We assume that each relation carries a distinct set of attributes. Furthermore, each attribute A_i in R comes equipped with a domain $dom(A_i)\subseteq\mathbb{U}$, where $\mathbb{U}$ is a countably infinite set. A tuple t of $R(A_1,\ldots,A_k)$ is simply an element of $dom(A_1)\times\cdots\times dom(A_k)$. A database of $\mathcal{R}$ is given by $D=(I_1,\ldots,I_n)$, where I_i is a finite set of tuples of R_i, for $i\in\{1,\ldots,n\}$. The active domain of D, denoted by $adom(D)$, is the set of elements from $\mathbb{U}$ present in D. Finally, the size $|D|$ of D refers to the number of tuples in D.

▷ *Distances and metric databases.* We further assume the presence of distance functions $\mathsf{d}_{A_i}: dom(A_i)\times dom(A_i)\to\mathbb{R}$, one for each attribute A_i in $\mathcal{R}$. A *metric database* (D,d) simply consists of a database D over $\mathcal{R}$ together with a collection of distance functions d_{A_i} for attributes A_i in $\mathcal{R}$.

To compare the distance between tuples on arbitrary sets X of attributes, we define $\mathsf{d}_X(s,t)=\max\{\mathsf{d}_{A_i}(s[A_i],t[A_i])\,|\,A_i\in X\}$, provided, of course, that s and t are defined over a set Y of attributes such that $X\subseteq Y$. For example, when dealing with numerical attributes A_i for which $d_{A_i}(s,t)=|s[A_i]-t[A_i]|$, we have that $\mathsf{d}_X(s,t)=\max\{|t[A_i]-s[A_i]|\,|\,A_i\in X\}$. We also need to lift distance functions to sets of tuples, *i.e.,* database instances. Given two sets C and D of tuples, we define for $s\in C$, $\mathsf{d}_X(s,D):=\min\{\mathsf{d}_X(s,t)|t\in D\}$ and $\mathsf{d}_X(C,D)=\max\{\max_{s\in C}\mathsf{d}_X(s,D),\max_{t\in D}\mathsf{d}_X(t,C)\}$. In addition, we define $\mathsf{diam}_X(D):=\max\{\mathsf{d}_X(s,t)|s,t\in D\}$. We denote the diameter with $\mathsf{diam}(D)$ when X is the set of all attributes in D.

Example 1. Consider a part D of the `Lineitem` table from the TPCH benchmark as shown in Figure 1 with attributes line number (LN), quantity (QT), extended price (EP), line status (LS), ship date (SD) and ship mode (SM). To turn D into a metric database (D,d) we equip each of the attributes with a distance function. For example, on the numerical attributes extended price and quantity we can use the absolute difference between values, on ship date one can use a date-specific distance function, and on the remaining categorical attributes the discrete distance function can be used. Consider tuples t_1 and t_2 in D. Their distance on the extended price attribute is then given by $\mathsf{d}_{\mathsf{EP}}(t_1,t_2)=|50634.87-11379.84|=39255.03$. Now if we include the quantity attribute when comparing these two tuples, then $\mathsf{d}_{\{\mathsf{EP},\mathsf{QT}\}}(t_1,t_2)=\max\{|50634.87-11379.84|,|39-8|\}=39255.03$. In this case the distance remains unchanged since the EP attribute dominates the distance value. As another example, using the discrete distance function on the line status attribute we have that $\mathsf{d}_{\mathsf{LS}}(t_1,t_2)=1$ whereas $\mathsf{d}_{\mathsf{LS}}(t_2,t_3)=0$.

	LN	QT	EP	LS	SD	SM
t_1:	1	39	50634.87	O	1997-04-12	REG AIR
t_2:	2	8	11379.84	F	1992-10-23	AIR
t_3:	3	32	53079.36	F	1994-04-23	RAIL
t_4:	4	12	22341.12	F	1993-08-11	REG AIR
t_5:	5	27	29542.86	F	1992-10-28	TRUCK
t_6:	6	11	16350.18	O	1997-11-28	REG AIR
t_7:	7	3	4065.99	O	1996-10-07	RAIL
t_8:	8	12	18102.24	O	1996-04-30	RAIL
t_9:	9	37	56625.91	O	1997-09-12	TRUCK
t_{10}:	10	22	39112.26	F	1992-12-24	RAIL

Fig. 1. An instance D of the `Lineitem` relation.

To illustrate how the distance between sets of tuples is measured, consider two subsets S_1 and S_2 of the instance D given by $S_1=\{t_1,t_3\}$ and $S_2=\{t_5,t_6\}$. For simplicity, we only use the quantity attribute. First, note that the distance between t_1 and S_2 is given by $\mathsf{d}_{\mathsf{QT}}(t_1,S_2)=\min\{\mathsf{d}_{\mathsf{QT}}(t_1,t_5),\mathsf{d}_{\mathsf{QT}}(t_1,t_6)\}=\min\{12,28\}=12$. Similarly, one can verify that $\mathsf{d}_{\mathsf{QT}}(t_3,S_2)=5$, $\mathsf{d}_{\mathsf{QT}}(t_5,S_1)=5$ and $\mathsf{d}_{\mathsf{QT}}(t_6,S_1)=21$. Then, the distance between S_1 and S_2 is given by $\mathsf{d}_{\mathsf{QT}}(S_1,S_2)=\max\{\max\{12,5\},\max\{5,21\}\}=21$. □

▷ *Conjunctive queries.* We consider conjunctive queries (CQ) specified by

$$Q_i(Y)=\pi_Y\sigma_{F_i}(R'_1\times\cdots\times R'_k),$$

where each R'_j is a renaming $\rho_j(R_j)$ of a relation R_j in $\mathcal{R}$, Y is a set of attributes and F_i is a selection predicate consisting of equality conditions of the form $A_i=A_j$ or $A_i=c$ for attributes A_i and A_j and constant $c\in dom(A_i)$.

3 The valid covering selection problem

In this section we first define the notion of a covering of a metric database relative to a set of attributes as a way of *approximating the data*. Next, we use these coverings to *approximate the results of queries* in some workload with a given budget on the available space to store the coverings of the data.

▷ *Data approximation by coverings.* Consider two metric databases (D,d) and (C,d) over $\mathcal{R}$ using the same distance functions d_{A_i}. Let X be a set of attributes and let $\epsilon\geqslant 0$. We say that (C,d) is an (X,ϵ)-*covering of* (D,d) if for any tuple $s\in D$ there exists a tuple $t\in C$ such that $\mathsf{d}_X(s,t)\leqslant\epsilon$, indicating that any tuple in D is close to a tuple in C relative to the set X of attributes. Furthermore, given a space budget constraint $\mathbb{B}$, we say that an (X,ϵ)-covering (C,d) of (D,d) is *valid relative* to $\mathbb{B}$, or simply *valid*, if $|C|\leqslant\mathbb{B}$, *i.e.,* the covering fits in the available space. A collection of coverings $(C_1,\mathsf{d}),\ldots,(C_\ell,\mathsf{d})$ of (D,d) is valid if $|C_1\cup\cdots\cup C_\ell|\leqslant\mathbb{B}$. Clearly, when valid coverings are concerned, the budget $\mathbb{B}$ imposes constraints on ϵ, and vice versa. For example, suppose that $\mathbb{B}=1$ then $\epsilon=\mathsf{diam}(D)$; if $\mathbb{B}=|D|$ then ϵ can be taken to be zero.

▷ *Query approximation by coverings.* We want to use coverings to approximate query answers. More specifically, we are given a space budget $\mathbb{B}$ and a query workload

$\mathcal{Q}=\{Q_1,\ldots,Q_\ell\}$ consisting of CQ queries. In addition, the frequency or importance of query Q_i in the workload is given by a parameter p_i. Then, given a metric database (D,d) we want to find the best valid collection of ℓ coverings (C_i,d), $i\in\{1,\ldots,\ell\}$, of (D,d) that can be used to approximate each query Q_i in $\mathcal{Q}$ relative to a user-defined set Z_i of attributes in the result schema of the query and accuracy bounds δ_i. Formally:

Valid Covering Selection Problem. *Given a metric database (D,d), query workload $\mathcal{Q}=\{Q_1(Y_1),\ldots,Q_\ell(Y_\ell)\}$, sets of attributes $Z_i\subseteq Y_i$, weights p_i, accuracy bounds δ_i, for $i\in\{1,\ldots,\ell\}$, and a budget constraint $\mathbb{B}$, find for each query Q_i a covering (C_i,d) of (D,d) such that $\max_{i\in\{1,\ldots,\ell\}} p_i\cdot|\delta_i-\mathsf{d}_{Z_i}(Q_i(D),Q_i(C_i))|$ is minimised and in addition, the collection $(C_1,\mathsf{d}),\ldots,(C_\ell,\mathsf{d})$ is valid relative to $\mathbb{B}$.* □

That is, this problem asks which coverings one should store in the available space as to "best" approximate the queries in the query workload. Here, with "best" we mean that $Q(C_i)$ approximates $Q(D)$ on the given attributes Z_i as close as possible to the user-defined accuracy bound δ_i. A naive approach for solving this problem is to just try all possible coverings and select the best ones. Not only is this exhaustive enumeration of coverings undesirable, it also requires the identification of coverings that are valid. Clearly, one cannot compute all such coverings efficiently. Furthermore, to select the best set of coverings one needs to compute $Q_i(D)$ (and also $Q_i(C)$ for that matter). Recall that we want to speed-up query evaluation by considering approximations. The exact computation of $Q_i(D)$ to find out the best way to approximate $Q_i(D)$ is clearly not an option! To make the valid covering selection problem feasible, one therefore needs to address the following two challenges.

Size estimation: We are looking for valid coverings. This implies that one must be able to determine the sizes of (X,ϵ)-coverings to identify those coverings that are valid, *i.e.,* fit into the budget. For this purpose we extend the catalog of the DBMS with information about valid coverings. We show in the next section how this can be efficiently implemented on top of a DBMS.

Error propagation: What can we say about $\mathsf{d}_Z(Q(D),Q(C))$ *without storing* (C,d) and *without evaluating* $Q(D)$ and $Q(C)$? That is, how does the accuracy bound on the data affect the accuracy bound on the query results? We will show in Section 5 that one can estimate $\mathsf{d}_Z(Q(D),Q(C))$ solely based on the structure of the query Q and the knowledge that (C,d) is an ϵ-covering.

4 Size estimation of coverings

In this section we consider how to estimate the size of an (X,ϵ)-covering of a metric database (D,d) for a given set X of attributes and accuracy value ϵ. The size of a minimum (X,ϵ)-covering is often referred to as the (X,ϵ)-*covering number* and will be denoted by $N(D,X,\epsilon)$. Not surprisingly, it is infeasible to compute $N(D,X,\epsilon)$ in practice. Indeed, one can verify that the computation of $N(D,X,\epsilon)$ corresponds to finding a solution to the VERTEX COVER problem, which is known to be NP-complete [2]. Although algorithms exist that approximate $N(X,D,\epsilon)$ (*e.g.*, based on [3]), they rely on efficient methods that, given a set S of tuples in D find a tuple t that maximises $d_X(t,S)$. Unfortunately, most database systems do not adequately support the indexing

of tuples relative to arbitrary distance functions; a crucial feature for finding farthest removed tuples. It is outside the scope of this paper to bring database systems up-to-date with recent advances in metric indexing techniques as reported in [4].

Instead we aim to expand the DBMS's catalog with quantitative information on (X,ϵ)-coverings for various sets X of attributes and accuracy values ϵ. We particularly want that this information is easy to compute and maintain within the DBMS. As a first attempt, one can use $\tilde{N}(D,X,\epsilon)=\mathsf{diam}_X(D)/\epsilon$ as a trivial upper bound on $N(D,X,\epsilon)$. Intuitively, this upper bound assumes uniform distribution of values in X. A more sensible upper bound is given by $\min\{\tilde{N}(D,X,\epsilon),|\pi_X(D)|\}$ since $|\pi_X(D)|$ also provides an upper bound on $N(D,X,\epsilon)$.

To obtain a more fine-grained, yet efficient-to-compute upper bound for $N(D,X,\epsilon)$, we further assume that the domain values of attributes in X can be (*e.g.,* lexicographically) sorted. We can then obtain an estimate for $N(D,X,\epsilon)$ by counting the number of non-empty buckets in an equi-width histogram $H(D,X,\epsilon)$. We denote this estimate by $\hat{H}(D,X,\epsilon)$ for a given histogram $H(D,X,\epsilon)$.

Recall that an equi-width histogram $H(D,X,\epsilon)$ consists of k tuples $t_1,\ldots,t_k$ such that (i) $t_1[X]<t_2[X]<\cdots<t_k[X]$; (ii) for each $i\in\{1,\ldots,k-1\}$, $\mathsf{d}_X(t_i,t_{i+1})=\epsilon$; and finally (iii) for $D_i=\{t\in D\,|\,t_i[X]\leqslant t[X]<t_{i+1}[X]\}$ we have $D=D_1\cup\cdots\cup D_k$. That is, $H(D,X,\epsilon)$ partitions the data into "buckets" D_i of diameter ϵ.

We next describe a procedure, referred to as Cover_Estim, for computing $\hat{H}(D,X,\epsilon)$. The pseudo-code of this procedure is shown in Figure 2. In a nutshell, Cover_Estim(D,X,q,ϵ) recursively processes the attribute list X (line 5). When the current attribute is A_i, the algorithm has already computed a histogram $H(D,\langle A_1,\ldots,A_{i-1}\rangle,\epsilon)$ consisting of $\hat{H}_{i-1}$ non-empty buckets and now further refines each of these buckets B in $H(D,\langle A_1,\ldots,A_{i-1}\rangle,\epsilon)$ by means of the sub-procedure bucket(B,A_i,ϵ). The result is a histogram $H(D,\langle A_1,\ldots,A_i\rangle,\epsilon)$ consisting of $\hat{H}_i$ buckets, where $\hat{H}_i$ is obtained from $\hat{H}_{i-1}$ by adding for each bucket B in $H(D,\langle A_1,\ldots,A_{i-1}\rangle,\epsilon)$ the number of buckets returned by bucket(B,A_i,ϵ). The recursive procedure halts when either an empty bucket is considered (line 2) or when enough attributes have been processed (line 7), at which point we return the trivial upper bound $\min\{\mathsf{diam}_{X'}(D)/\epsilon,|\pi_{X'}(D)|\}$ for the remaining attributes $X'=\langle A_{q+1},\ldots,A_p\rangle$. Note that q is a user-chosen parameter that determines how many attributes should be processed using the bucket refinement procedure. If $q=p$, the recursion stops when all attributes have been processed. In this case, a recursive call with $X=\emptyset$ is made, indicating that the non-empty bucket B under consideration does not need any further refinement and thus will contribute a count of 1 to the estimate (line 8). Also note that if $q=0$ then no recursion takes place and the naive upper bound $\min\{\mathsf{diam}_X(D)/\epsilon,|\pi_X(D)|\}$ is returned for the complete attribute set X (line 7).

It remains to detail the sub-procedure bucket(D,A_i,ϵ) which, given a database D, attribute A_i and accuracy value ϵ, constructs a partition $D_1\cup\cdots\cup D_k$ of the data corresponding to a histogram $H(D,A_i,\epsilon)$. Assuming that the sorted attribute values for A_i can be cast as numerical values, which is often the case in practice, we can leverage the presence of the `Width_bucket` function in `SQL`. This function takes as input an attribute A_i, the minimal and maximal value of the active domain of A_i in D, and a desired number of buckets. The result consists of pairs (t,i) where $t\in D$ and i is the

Procedure Cover_Estim

Input: A database D, list of attributes $X=\langle A_1,A_2,\ldots,A_p\rangle$, number of attributes to be processed by the non-naive method q, accuracy threshold ϵ.

Output: Number $\hat{H}(X,D,\epsilon)$ of non-empty buckets in equi-width histogram $H(X,D,\epsilon)$.

1. $\hat{H}:=0$;
2. **if** $D=\emptyset$ **then return** 0;
3. **if** $p>0$ **then**
4. **if** $q>0$ **then**
5. **for each** $B=\mathsf{bucket}(D,A_1,\epsilon)$ **do** $\hat{H}:=\hat{H}+\mathsf{Cover\_Estim}(B,\langle A_2,\ldots,A_p\rangle,\epsilon,q-1)$;
6. **return** $\hat{H}$.
7. **else return** $\min\{\mathsf{diam}_X(D)/\epsilon,|\pi_X(D)|\}$.
8. **else return** 1.

Fig. 2. Procedure for estimating the size of a covering.

unique bucket number to which t belongs. From this, the number of non-empty buckets and a histogram can easily be computed. More specifically, $\mathsf{bucket}(D,A_i,\epsilon)$ can be implemented by means of the following SQL expression:

```
SELECT Width_bucket(A_i,min,max,min{⌈diam_{A_i}(D)/ε⌉,|π_{A_i}(D)|}) AS bucket
FROM R GROUP BY bucket ORDER BY bucket
```

where we use the estimate $\min\{\lceil\mathsf{diam}_{A_i}(D)/\epsilon\rceil,|\pi_{A_i}(D)|\}$ for an upper bound on the number of buckets. Furthermore, it should come as no surprise that the recursive procedure Cover_Estim can be implemented entirely in SQL, provided of course that we know the attributes in X up front (otherwise a recursive SQL query is needed). Indeed, the SQL implementation of Cover_Estim consists of nested variants of the SQL query given above, where the nesting level is determined by the number of attributes in X. Observe that Cover_Estim not only computes size bounds but returns actual coverings.

Example 2. Recall the `Lineitem` database D given in Example 1. For the extended price (EP) attribute the diameter of D is simply given by $\mathsf{diam}_{\mathsf{EP}}(D)=\max_{t\in D}t[\mathsf{EP}]-\min_{t\in D}t[\mathsf{EP}]=52559.92$. Observe that $|\pi_{\mathsf{EP}}(D)|=10$, hence at most 10 buckets are needed to exactly cover D on attribute EP. Taking our $\epsilon=8770$, the quantity $\min\{\lceil\frac{52559.92}{8770}\rceil,10\}=6$ is an upper bound on the number of buckets needed.

We now illustrate the Cover_Estim procedure. Firstly consider the evaluation of $\mathsf{Cover\_Estim}(D,\mathsf{EP},0,8770)$. In this case, all attributes are processed by the naive method ($q=0$) and therefore the procedure outputs the upper bound of 6 buckets. Next, we set $q=1$ so that the procedure $\mathsf{Cover\_Estim}(D,\mathsf{EP},1,8770)$ now uses the non-naive method by evaluating the above SQL query. We obtain a covering $C=\{t_1,t_4,t_6,t_7,t_{10}\}$ of D of size 5 buckets as follows: $B_1=\{t_7,t_2\},B_2=\{t_6,t_8\},B_3=\{t_4,t_5\},B_4=\{t_{10}\}$ and $B_5=\{t_1,t_3,t_9\}$. Tuples are sorted in each bucket. It is readily verified that C is an $(\mathsf{EP},8770)$-covering of D. From this small example we can already see that the non-naive method improves on the trivial upper bound on the number of buckets (five buckets rather than six). Furthermore we also note that C is an $(\{\mathsf{EP},\mathsf{QT}\},8770)$-covering of D. This is so because after the Cover_Estim procedure processes the EP attribute, the buckets do not require any further refinement. Indeed, for each bucket B_i, for $i\in\{1,\ldots,5\}$, we already have that $\mathsf{diam}_{\mathsf{QT}}(B_i)\leqslant\epsilon$. □

Remarks. (1) We described a very specific method for estimating $N(D,X,\epsilon)$. However, any other method (*e.g.*, based on k-means clustering or other kinds of histograms) can be easily plugged into our query approximation system. (2) It is important to observe that the cost of estimating $N(D,X,\epsilon)$ is a one-time cost and can be done when the DBMS is idle. (3) Clearly, the order in which the attributes in X are fed to $\mathsf{Cover\_Estim}(D,X,\epsilon)$ directly impacts the estimate of $N(D,X,\epsilon)$. Further investigation is required to determine heuristics to select the best order.

5 Error propagation

The second challenge that we have to address is the efficient estimation of $\mathsf{d}_Z(Q(C),Q(D))$ for (X,ϵ)-coverings (C,d) of (D,d). That is, we need to estimate the error on the query result due to the use of coverings rather than the original database. Since our aim is to speed-up the query evaluation of Q, one cannot rely on computing $Q(D)$, $Q(C)$ and $\mathsf{d}_Z(Q(D),Q(C))$, as this requires evaluating the queries. The estimation procedure for $\mathsf{d}_Z(Q(D),Q(C))$ should thus be *independent* of (D,d) and thus also of the chosen covering (C,d). This bears the question whether the knowledge of the query Q and the fact that there is an (X,ϵ)-covering of the data is sufficient to obtain an accuracy bound on the query result. We answer this question affirmatively, provided that we slightly relax the query Q into an approximate query $\widetilde{Q}$, as will be explained shortly.

The overall strategy to estimate $\mathsf{d}_Z(Q(D),Q(C))$ then consists of showing under which conditions an (X,ϵ)-covering (C,d) of (D,d) can be transformed in a (Z,ϵ')-covering $\widetilde{Q}(C)$ of $Q(D)$. Given this, we can then estimate $\mathsf{d}_Z(Q(D),Q(C))$ using $\mathsf{d}_Z(Q(D),\widetilde{Q}(C))=\epsilon'$. In particular, we show how ϵ' can be expressed in terms of ϵ, hereby alleviating the need for evaluating any query in the estimation process.

$\triangleright$ *Query relaxation.* Let us first explain why query relaxations are needed. Suppose that $Q=\sigma_{A=a}(R)$ and let (D,d) be a metric database. Then $Q(D)$ contains all tuples $t\in D$ with $t[A]=a$. Take any (A,ϵ)-covering (C,d) of (D,d). Then, unless C contains tuples t of D with $t[A]=a$, we have that $Q(C)=\emptyset$ and thus $Q(C)$ is not a covering of $Q(D)$. In other words, we cannot guarantee that any (A,ϵ)-covering (C,d) of (D,d) suffices to approximate $Q(D)$. A similar situation arises when considering $Q=\sigma_{A=B}(R)$.

To remedy this situation, we not only approximate the data but also consider *relaxations* of queries. More specifically, we compare $Q=\sigma_{A=a}(R)$ on (D,d) with its relaxation $\widetilde{Q}=\sigma_{\mathsf{d}(A,a)\leqslant\eta}(R)$ on (C,d), for some value η. The semantics of $\widetilde{Q}$ is as expected: $\sigma_{\mathsf{d}(A,a)\leqslant\eta}(C)$ selects all tuples t in C for which $\mathsf{d}_A(t[A],a)\leqslant\eta$. When considering (A,ϵ)-coverings (C,d) of (D,d) with $\epsilon\leqslant\eta$, we then have that $\mathsf{d}_A(Q(D),\widetilde{Q}(C))\leqslant\epsilon$. Similarly, one can verify that when $Q=\sigma_{A=B}(R)$ is relaxed to $\widetilde{Q}=\sigma_{\mathsf{d}(A,B)\leqslant\eta}(R)$, then $\mathsf{d}_{A,B}(Q(D),\widetilde{Q}(C))\leqslant\epsilon$ for any $(\{A,B\},\epsilon)$-covering (C,d) of (D,d) with $\epsilon\leqslant\frac{\eta}{2}$. Note that for a selection condition $A=B$ to make sense, attributes A and B must have the same domain and distance function. We denote $\mathsf{d}_A=\mathsf{d}_B$ by d. Hence, $\sigma_{\mathsf{d}(A,B)\leqslant\eta}(C)$ selects all tuples t in C such that $\mathsf{d}(t[A],t[B])\leqslant\eta$. Now, given a tuple t in D such that $t[A]=t[B]$ and given an $(\{A,B\},\epsilon)$-covering (C,d) of (D,d) with $\epsilon\leqslant\frac{\eta}{2}$, we have that there exists a tuple $t'\in C$ such that $\mathsf{d}_{A,B}(t,t')\leqslant\epsilon$. Indeed,

observe that $\mathsf{d}(t'[A],t'[B]) \leqslant \mathsf{d}(t[A],t'[A]) + \mathsf{d}(t[B],t'[B]) = 2\epsilon \leqslant \eta$. Hence, $t' \in \widetilde{Q}(C)$ and $\widetilde{Q}(C)$ is a covering of $Q(D)$.

For a CQ query Q we denote by $\widetilde{Q}_\eta$ the query obtained by replacing any selection predicate in Q by its relaxed version, *i.e.,* all occurrences of $\sigma_{A=a}$ and $\sigma_{A=B}$ in Q are replaced by $\sigma_{\mathsf{d}(A,a)\leqslant\eta}$ and $\sigma_{\mathsf{d}(A,B)\leqslant\eta}$, respectively.

Example 3. Recall again our `Lineitem` database D from Example 1. Consider the constant selection query Q which selects all tuples $t \in D$ with $t[\mathsf{EP}] = 18102.24$ and projects on the EP attribute. We reuse our covering from Example 2, *i.e.,* $C = \{t_1, t_4, t_6, t_7, t_{10}\}$. The relaxed query $\widetilde{Q}$ selects all tuples $t \in C$ with $t[\mathsf{EP}] \in [18102.24 - 8770, 18102.24 + 8770]$ and also projects on the EP attribute. Evaluating both queries, we have $Q(D) = \{t_8\}$ and $\widetilde{Q}(C) = \{t_4, t_6\}$, from which we obtain

$$\mathsf{d}_{\mathsf{EP}}(Q(D), \widetilde{Q}(C)) = 4238.88 \leqslant 8770.$$

Hence, $\tilde{Q}(C)$ is $(\mathsf{EP}, 8770)$-approximation of $Q(D)$. □

▷ *Propagation algorithm.* We next provide an algorithm, Error_Prop, that given a CQ query Q, a set X of attributes in $\mathcal{R}$, an accuracy value ϵ, and relaxation parameter η for $\widetilde{Q}_\eta$, returns:

(a) a set of attributes $\mathsf{prop}(Q, X)$ in the result schema of Q; and
(b) an error bound $\mathsf{err}(Q, \epsilon)$,

such that for *any* (X, ϵ)-covering (C, d) of (D, d) it is guaranteed that

$$\mathsf{d}_Z(Q(D), \widetilde{Q}_\eta(C)) \leqslant \mathsf{err}(Q, \epsilon)$$

for any non-empty $Z \subseteq \mathsf{prop}(Q, X)$. The algorithm Error_Prop works inductively on the structure of the query Q and is described in Figure 3. Its correctness can be readily verified but this is omitted due to space limitations.

▷ *Query column sets and error guarantees.* One can see from the description of $\mathsf{Error\_Prop}(Q, X, \epsilon, \eta)$ in Figure 3 that certain conditions on coverings need to hold when using them to approximate $Q(D)$. That is, when $\mathsf{prop}(Q, X)$ is empty, insufficiently many attributes are covered to approximate Q. Observe that $\mathsf{prop}(Q, X)$ is empty when X does not contain (i) any attribute in the relations occurring in Q (line 3); (ii) an attribute that appears in a selection condition (lines 8, 11); or (iii) any of the projected attributes in Q (line 14). Given a set of attributes Z in the result schema of Q, one can compute for each relation R_i in $\mathcal{R}$ the minimal set of attributes X_i such that for $X = X_1 \cup \cdots \cup X_n$, $Z \subseteq \mathsf{prop}(Q, X)$. In other words, the X_i's are the attributes that are required to be covered in R_i in order to approximate Q on Z. We denote by $\mathsf{qcs}(Q, Z)$ the set of pairs (R_i, X_i). In analogy with the BlinkDB system, we also refer to $\mathsf{qcs}(Q, Z)$ as the *query column set of Q relative to Z*. The query column sets can be computed by starting from Z and by reversely applying the different cases in $\mathsf{Error\_Prop}(Q, X, \epsilon, \eta)$ for $\mathsf{prop}(Q, X)$. We omit the details due to space limitations.

Furthermore, even when the query column set $X = \mathsf{qcs}(Q, Z)$ is covered it may be that $\mathsf{err}(Q, X) = +\infty$ and thus no approximation is achieved. This happens when ϵ is

Procedure Error_Prop (Q,X,ϵ,η)

1. **switch**
2. **case** $Q=R(A_1,\ldots,A_k)$
3. **return** $\mathsf{prop}(Q,X):=X\cap\{A_1,\ldots,A_k\}$; and $\mathsf{err}(Q,\epsilon):=\epsilon$;
4. **case** $Q=\rho\big(Q'(A_1,\ldots,A_k)\big)$ for some renaming $\rho=(A_1\mapsto B_1,\ldots,A_k\mapsto B_k)$
5. **return** $\mathsf{prop}(Q,X):=\rho\big(\mathsf{prop}(Q',X)\big)$; and $\mathsf{err}(Q,\epsilon):=\mathsf{err}(Q',\epsilon)$;
6. /* Where $\mathsf{prop}(Q',X)$ and $\mathsf{err}(Q',\epsilon)$ are the result of $\mathsf{Error\_Prop}(Q',X,\epsilon,\eta)$ */
7. **case** $Q=\sigma_{A=a}(Q')$
8. **return** $\mathsf{prop}(Q,X):=\mathsf{prop}(Q',X)$ if $A\in\mathsf{prop}(Q',X)$ and $\mathsf{prop}(Q,X):=\emptyset$ other-
9. wise; and $\mathsf{err}(Q,\epsilon):=\mathsf{err}(Q',\epsilon)$ if $\epsilon\leqslant\eta$ and $\mathsf{err}(Q,\epsilon):=+\infty$ otherwise;
10. **case** $Q=\sigma_{A=B}(Q')$
11. **return** $\mathsf{prop}(Q,X):=\mathsf{prop}(Q',X)$ if $A,B\in\mathsf{prop}(Q',X)$ and $\mathsf{prop}(Q,X):=\emptyset$ other-
12. wise; and $\mathsf{err}(Q,\epsilon):=\mathsf{err}(Q',\epsilon)$ if $\epsilon\leqslant\eta/2$ and $\mathsf{err}(Q,\epsilon):=+\infty$ otherwise;
13. **case** $Q=\pi_Y(Q')$
14. **return** $\mathsf{prop}(Q,X):=\mathsf{prop}(Q',X)\cap Y$; and $\mathsf{err}(Q,\epsilon):=\mathsf{err}(Q',\epsilon)$;
15. **case** $Q=Q_1\times Q_2$
16. **return** $\mathsf{prop}(Q,X):=\mathsf{prop}(Q_1,X)\cup\mathsf{prop}(Q_2,X)$; and $\mathsf{err}(Q,\epsilon):=\max\{\mathsf{err}(Q_1,\epsilon),$
17. $\mathsf{err}(Q_2,\epsilon)\}$.
18. /* Here, $\mathsf{prop}(Q_i,X)$ and $\mathsf{err}(Q_i,\epsilon)$ are given by $\mathsf{Error\_Prop}(Q_i,X,\epsilon,\eta)$, for $i=1,2$.*/

Fig. 3. Error propagation algorithm.

too large compared to the chosen relaxation parameter η and when Q contains selection conditions (lines 9, 12). In particular, from $\mathsf{Error\_Prop}(Q,X,\epsilon,\eta)$ we obtain the following error guarantees:

$$\text{If } \left\{\begin{array}{l}\text{(a) } Q \text{ does not contain selection conditions}\\ \text{or}\\ \text{(b) } Q \text{ only contains constant selection conditions and } \epsilon\leqslant\eta\\ \text{or}\\ \text{(c) } Q \text{ contains an equality selection conditions and } \epsilon\leqslant\eta/2\end{array}\right\}\Longrightarrow \mathsf{err}(Q,\epsilon)=\epsilon.$$

Otherwise, we have an unbounded $(+\infty)$ error. Of course, this also implies that unbounded errors can be avoided altogether by considering relaxations $\widetilde{Q}_\eta$ that depend on ϵ, *i.e.,* by letting $\eta=0$ in case (a); $\eta=\epsilon$ in case (b); and $\eta=2\epsilon$ in case (c). In the following, we always assume that these relaxations are used when approximating Q and simply denote the relaxation by $\widetilde{Q}$.

Example 4. Consider query Q' obtained from modifying query Q given in Example 3 by projecting on two attributes $\{\mathsf{EP},\mathsf{QT}\}$ instead of only projecting on EP. Suppose that we want to approximate Q' on these attributes, *i.e.,* $Z=\{\mathsf{EP},\mathsf{QT}\}$ with user-defined threshold $\eta=8770$. To identify which (X,ϵ)-coverings of D can be used to approximate Q', we must have that $Z\subseteq\mathsf{prop}(Q',X)$. Evaluating the procedure $\mathsf{Error\_Prop}$ tells us that any (X,ϵ)-covering of D such that $\{\mathsf{EP},\mathsf{QT}\}\subseteq X$ will do. This follows directly from the selection (line 8) and projection (line 14) rule in the procedure. Consider the $(\{\mathsf{EP},\mathsf{QT}\},8770)$-covering C from Example 2. It is readily verified that for $\epsilon=8770$, we get an error bound $\mathsf{err}(Q',\epsilon)=\epsilon=8770$. Hence, for any $Z\subseteq\{\mathsf{EP},\mathsf{QT}\}$ we guarantee that $\mathsf{d}_Z(Q'(D),\widetilde{Q'}_\eta(C))\leqslant 8770$, *i.e.,* we can approximate Q' within the user-defined threshold η on attributes Z. □

6 Valid covering selection

We now have all ingredients at hand to describe our approach for solving the valid covering selection problem. Let $\mathcal{Q}=\{Q_1,\ldots,Q_\ell\}$ be the query workload consisting of ℓ CQ queries. For each query Q_i the user specifies its importance p_i, the set of attributes Z_i in Q_i's result schema to be approximated, and desired error bound δ_i. Furthermore, a space budget $\mathbb{B}$ is given. Our approach works in four steps:

1. We collect the query column sets $\mathsf{qcs}(Q_i,Z_i)$, for $i\in\{1,\ldots,\ell\}$. Recall that in order to approximate $Q_i(D)$ on attributes Z_i, one minimally needs to cover all attributes in $\mathsf{qcs}(Q_i,Z_i)$.
2. Next, we inspect the DBMS catalog that, by using the size estimation method described in Section 4, is now extended with quadruples (R_j,X_j,ϵ_j,N_j), indicating that there is an (X_j,ϵ_j)-covering of size N_j of the instance I_j of R_j. Note that there may be multiple coverings on each relation. Denote by $\mathsf{cov}(D)$ the collection of all such quadruples in the catalog. Clearly, considering all possible coverings on $\mathcal{R}$ would lead to exponentially many coverings in $\mathsf{cov}(D)$. Instead, we assume that a set of candidate covering attributes is provided based on those that actually were needed in the past or simply by inspecting the query column sets of the workload queries.
3. We then solve a mixed integer linear program (MILP). Part of the solution of this program are variables x_{ij} that, when set to 1, indicate that the i^{th} covering in $\mathsf{cov}(D)$ is used to approximate Q_j.
4. We materialise all coverings $(C_1,\mathsf{d}),\ldots,(C_\ell,\mathsf{d})$ identified in the previous step, hereby avoiding replicating the same covering on a relation. Just like in BlinkDB, the materialisation step is a one-time cost and if the query workload $\mathcal{Q}$ is representative for the past, present and future workload, the stored coverings can be used for any future incoming queries as well. For now, to obtain an approximation for queries in $\mathcal{Q}$ we evaluate $\widetilde{Q_j}(C_j)$ on the stored coverings. Recall that $\widetilde{Q_j}$ is the relaxation of Q_j by setting η to the appropriate value 0, ϵ_j, or $2\epsilon_j$, where ϵ_j is the accuracy of covering (C_j,d_j) (See Section 5).

The MILP ensures that (i) all coverings fit into the available space budget; and (ii) the best possible accuracies of these coverings are selected for approximating the workload queries. Observe that, assuming that $\mathsf{cov}(D)$ is available, we only need to evaluate the relaxed queries on coverings. No other query evaluation or access to the data is needed. This implies, among other things, that the size of the MILP is independent of the size of D and that solving it is a cost that is negligible. We verify this in the experimental section. It remains to detail the mixed integer linear program.

▷ *MILP formulation.* Part of the MILP consists of a simple set covering problem: for each $Q_i\in\mathcal{Q}$ find $(R_1,X_1,\epsilon_1,N_1),\ldots,(R_n,X_n,\epsilon_n,N_n)$ in $\mathsf{cov}(D)$ that cover $\mathsf{qcs}(Q_i,Z_i)$. More specifically, if $\mathsf{qcs}(Q_i,Z_i)=\{(R_1,Y_1),\ldots,(R_n,Y_n)\}$ then we must have that $Y_i\subseteq X_i$ for $i\in\{1,\ldots,n\}$. We encode this set cover problem in the MILP in the standard way. Let $I=\{1,\ldots,|\mathsf{cov}(D)|\}$ and $J=\{1,\ldots,\ell\}$. For each $(i,j)\in I\times J$ and relation name $R\in\mathcal{R}$, we introduce a constant c_{ij}^R and boolean variable x_{ij}^R. Here, $c_{ij}^R=1$

if the i^{th} covering in $\mathsf{cov}(D)$ contains the attributes in $\mathsf{qcs}(Q_j,Z_j)$ corresponding to R; and $c_{ij}^R=0$ otherwise. Furthermore, $x_{ij}^R=1$ is to indicate that this covering on R is used to approximate Q_j on the Z_j attributes and $x_{ij}^R=0$ indicates the opposite. To ensure that $\mathsf{qcs}(Q_j,X_j)$ is fully covered we thus require that

$$\sum_{i\in I} c_{ij}^R x_{ij}^R \geqslant 1 \quad \begin{array}{l}\text{(for each } j\in J,\ R\in\mathcal{R} \text{ such that } (R,X)\in\mathsf{qcs}(Q_j,Z_j)\\ \text{for some non-empty set } X \text{ of attributes.)}\end{array}$$

In addition, all coverings in $\mathsf{cov}(D)$ that are used for approximating queries in $\mathcal{Q}$ need to be stored and must fit within the available space budget $\mathbb{B}$. For each $i\in I$ we therefore introduce a variable x_i that will be set to 1 if any of the x_{ij}^R's for $j\in J$ is 1; and x_i is set to 0 otherwise. In other words, $x_i=\max\{x_{ij}^R \mid j\in J, R\in\mathcal{R}\}$ and indicates which coverings in $\mathsf{cov}(D)$ are being used. We therefore require

$$x_{ij}^R \leqslant x_i \quad \text{(for each } i\in I,\ j\in J,\ R\in\mathcal{R}\text{), and } x_i \leqslant \sum_{j\in J, R\in\mathcal{R}} x_{ij}^R \quad \text{(for each } i\in I\text{).}$$

The space budget constraint is simply given by $\sum_{i\in I} x_i N_i \leqslant \mathbb{B}$, where N_i is the size of the i^{th} covering in $\mathsf{cov}(D)$. It remains now to relate the selected coverings (*i.e.,* those with $x_{ij}^R=1$) to the error bound on the corresponding query Q_j. Since this error is given by $\mathsf{err}(Q_j,\epsilon)=\epsilon$ we need to determine the maximum ϵ used in the coverings for Q_j. For this purpose we introduce variables y_j, for $j\in J$, and require that

$$x_{ij}^R c_{ij}^R \epsilon_i \leqslant y_j \quad \text{(for each } i\in I,\ j\in J,\ R\in\mathcal{R}\text{)}$$

where ϵ_i is the accuracy value of the i^{th} covering in $\mathsf{cov}(D)$. Finally, the objective function of the MILP is

$$\textbf{minimise:} \quad \max_{j\in J} p_j|\delta_j - y_j|,$$

where δ_j is the user-specified accuracy threshold. It is easily verified that the objective function can be transformed into a linear constraint, *i.e.,* without using $\max$ and absolute value $|\cdot|$.

Example 5. We next illustrate the interaction of the space budget and accuracy threshold in the valid covering selection method. Recall query Q' from Example 4 and the $(\{\mathsf{EP},\mathsf{QT}\},8770)$-covering, here denoted by C_1, from example 2. Let C_2 be another covering of our `Lineitem` database D consisting of three buckets, *i.e.,* $C_2=\{t_4,t_7,t_{10}\}$. It is readily verified that C_2 is a $(\{\mathsf{EP},\mathsf{QT}\},17520)$-covering. Then we have $\mathsf{cov}(D)$ with two coverings $\{C_1,C_2\}$ of size 5 and 3 respectively. We know that $\mathsf{qcs}(Q',Z)=\{\mathsf{EP},\mathsf{QT}\}$. Let $p_1=1$, $\delta_1=5500$ and $\mathbb{B}=4$.

The MILP can now be formulated with the required parameters as above. In this case the program has a simple task: to decide which of the two coverings should be used to approximate Q'. Based on the constraints, an optimal solution would set x_{21} to 1, *i.e.,* covering C_2 is chosen since only C_2 fits into $\mathbb{B}$. Let us consider another scenario where more space is available and we increase our space budget, *i.e.,* $\mathbb{B}=7$. Now, we see that any of the two coverings can fit into the available budget. Note that the user desired error bound remains unchanged. Again, by looking at the objective function of the MILP, it is easy to see that an optimal solution would set x_{11} to 1, *i.e.,* covering C_1 is chosen because the propagated error for covering C_1, $(\epsilon_1=8770)$, is more close to the desired error bound, δ_1 than the propagated error of C_2, $(\epsilon_2=17520)$. □

7 Experimental Evaluation

In this section, we evaluate the performance of the procedure Cover_Estim for estimating the size and computing the coverings of the data, and the accuracy of our solution to the valid selection problem on individual queries and on queries in some workload.

▷ *Evaluation setting.* Our experiments were run on a GNU/Linux machine with Intel(R) Xeon(R) CPU 2.90 GHZ (16 Cores) and 32 GB memory. We use PostgreSQL as the underlying database system. All experiments were repeated five times and averages are reported. We used two datasets: (i) the TPC-H benchmark data [1] (scale factor 1) consisting of 9 million tuples (1GB); and (ii) the Big Data benchmark [2] (scale factor 1) consisting of tables `uservisits` and `rankings` of 8 million (1.28GB) and 155 million (25.4GB) tuples, respectively. For schema details on the datasets, consult the links in the footnote. On the TPC-H data, our query workload $\mathcal{Q}$ consists of variants of Q_1, Q_3, Q_6, Q_{13}, Q_{19} of the TPC-H queries. For the Big Data benchmark data, we use a variant of their scan and aggregate query. In all of our experiments we consider coverings on attributes that have a sorted domain and use distance functions as described in Section 2.

▷ *Covering computations.* We first experimentally validate the efficiency of the procedure Cover_Estim, as described in Section 4, then illustrate how different datasets can be compressed by means of coverings, and finally investigate the impact of the parameter q on the quality and efficiency of the bounds returned by Cover_Estim.

Figure 4 shows the time to compute the size of coverings and the time to materialise them on the Big Data benchmark dataset. More specifically, we varied (i) the sizes of the input tables `uservisits` (1000k to 10000k) tuples and `rankings` (1000k to 10000k) tuples; and (ii) the sets X of attributes to be covered. On the left, we report the times for individual attributes *duration* and *adrevenue* in `uservisits`, and attributes *avgduration* and *pagerank* in `rankings`; on the right we consider the combined attribute sets $\{$*duration*, *adrevenue*$\}$ and $\{$*avgduration*, *pagerank*$\}$. We fixed ϵ to be 0.0001.

Not surprisingly, estimating the size of coverings requires considerably less time than materialising them. Indeed, while the size estimation typically takes a couple of seconds, the materialisation takes tens of seconds. This verifies our claim that extending the DBMS's catalog with quantitative information on coverings is feasible, especially since this is a one-time cost and can be computed when the system is idle. We further observe that the running times strongly depend on the set X of attributes. In particular, the running time increases when X consists of more attributes. This is not unexpected since a larger X results in a larger (X,ϵ)-covering. A similar behaviour is observed when varying ϵ, *i.e.,* the smaller the ϵ, the more time it takes to bound the size of the coverings. Experiments on the TPC-H data gave analogous results (not reported).

We next considered the compressibility of the datasets. Figures 5 (a) and (b) show the size of the resulting covering on the two tables in the Big Data benchmark data set for varying values of ϵ and for the attribute sets considered earlier. We fixed the size of both tables to 10 million tuples. Similarly, Figure 5 (c) shows the size of coverings on the `lineitem` table in the TPC-H data set for the following sets of attributes:

[1] http://www.tpc.org/tpch/

[2] https://amplab.cs.berkeley.edu/benchmark/

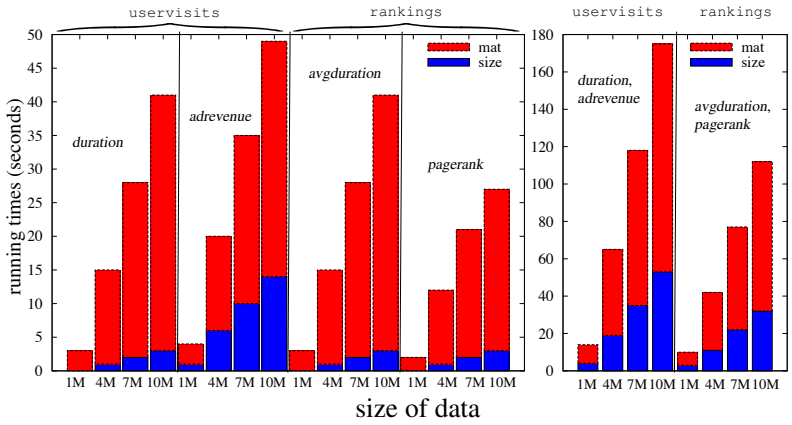


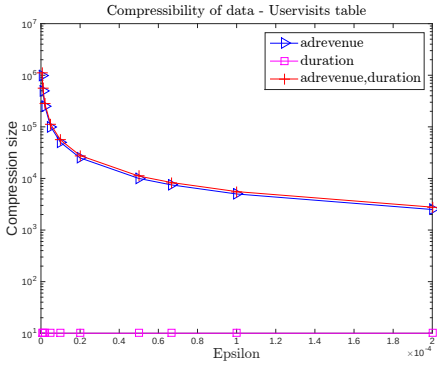
Fig. 4. Efficiency of Cover_Estim for computing size and materialisation of coverings.

{*lextendedprice*}, {*lextendedprice*, *lquantity*} and {*lextendedprice*, *lquantity*, *llinestatus*}. One can see that the datasets compress rather well: for reasonable values of ϵ the size of the corresponding covering provides a considerable reduction compared to the size of the original data. In other words, even for a small space budget $\mathbb{B}$ one can find accurate coverings of the data. As before, the more attributes are used in the covering, the larger the covering.

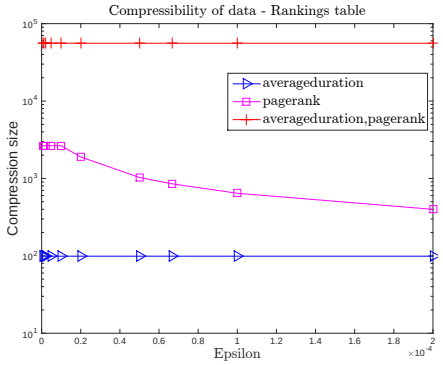
Finally, Figure 5 (d) shows the impact of the parameter q in Cover_Estim. Recall from Section 4 that q indicates how many attributes are processed by the recursive bucketisation process, and consequently, how many attributes are treated by the naive upper bound. Figure 5 (d) reports the effect on the `lineitem` table and attribute set {*lextendedprice*, *lquantity*, *ldiscount*}. We let $q{=}3$ (most fine-grained upper bound), $q{=}2$ (last attribute is treated by naive upper bound), and $q{=}1$ (last two attributes are estimated by naive upper bound). Not surprisingly, the quality of the size estimate degrades with decreasing q. On the other hand, the running times for Cover_Estim decrease when more attributes are treated by the naive upper bound. Indeed, our experiments (not reported due to space limitations) show that the size estimation for $q{=}2$ takes half the time when compared to $q{=}3$.

▷ *Query approximation and valid covering selection.* Our next set of experiments concerns the use of coverings to approximate query results. We first consider individual queries and compare the theoretical upper bound with the actual error made by our query approximation method. Next, we consider a query workload and investigate our solution to the valid cover selection problem.

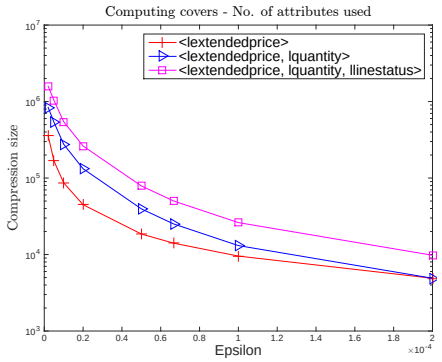
Figure 6 shows the comparison of the actual error $\mathsf{d}_Z(Q(D),\widetilde{Q}(C))$ with the theoretical upper bound $\mathsf{err}(Q,\epsilon)$ given in Section 5, for varying sizes of coverings C of D. More specifically, we express the size $|C|$ of C as a percentage of the size $|D|$ of D and report the error $\mathsf{err}(Q,\epsilon)$, where ϵ is the accuracy associated with the covering C. The computation of $\mathsf{d}_Z(Q(D),\widetilde{Q}(C))$ is done by evaluating $Q(D)$ and $\widetilde{Q}(C)$ and by computing the distance between them. Due to space limitations we only report two settings. In Figure 6 (a) we consider the `lineitem` table of the TPC-H data and a con-



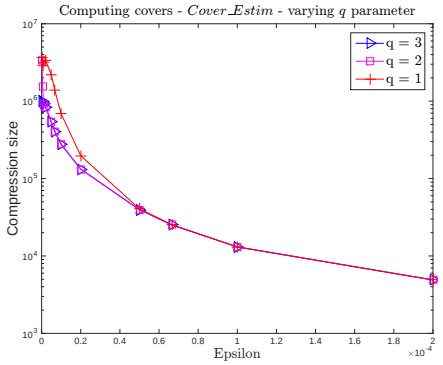
(a) Big Data benchmark: Compressibility



(b) Big Data benchmark: Compressibility



(c) TPC-H data: Attribute size effect



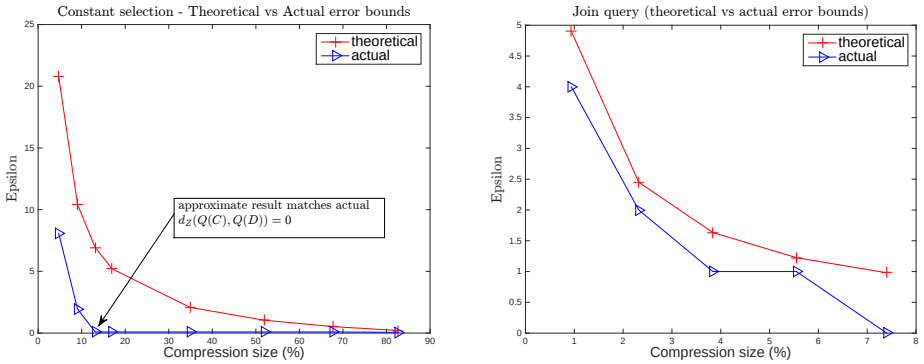
(d) TPC-H data: Parameter q effect

Fig. 5. Compressibility of datasets and impact of parameter q.

stant selection query Q on the *lextendedprice* attribute. Coverings are on this attribute only. In Figure 6 (b) we consider a join query Q (*i.e.,* cartesian product followed by equality selection) involving both tables in the Big data benchmark data. Coverings are on attribute sets $\{$*duration*, *adrevenue*$\}$ on `uservisits` and $\{$*avgduration*, *pagerank*$\}$ on `rankings`.

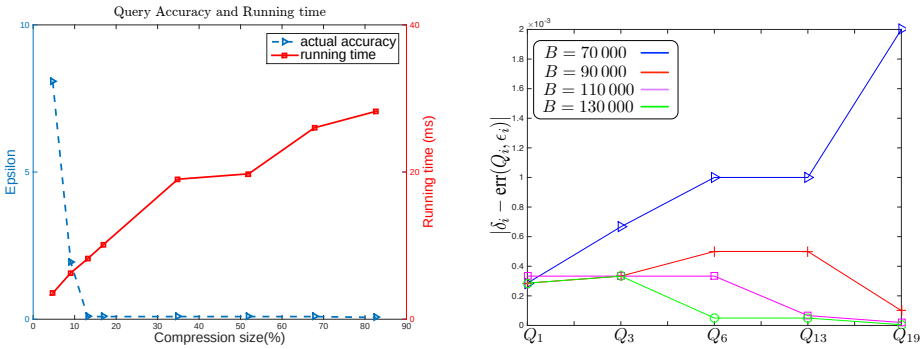
These experiments show that our approach actually provides better actual accuracy bounds on the query results than is anticipated by the theoretical upper bound. In particular, we note that for highly compressible attributes, such as *lextendedprice* in `lineitem` we can get an actual error of 0 using only a small fraction (13.2%) of the original dataset. For this particular setting, Figure 7 (a) verifies that answering queries on coverings takes less time than when using the original data and, more importantly, that the error made by the approximation is within reasonable bounds. In particular, Figure 7 (a) shows the actual error for the constant selection query on the TPC-H data for various sizes of coverings and the time it takes to answer its relaxation on the coverings. For example, evaluating the relaxation using 13.2% of the data takes $1/4$ of the time needed to evaluate the query on the original data without loss of accuracy.

Finally, we consider a query workload $\mathcal{Q}$ of the 5 TPC-H queries mentioned earlier. We extended the DBMS catalog with 50 different coverings (some of them over the



(a) TPC-H data: accuracy (b) Big Data benchmark: accuracy

Fig. 6. Comparison of actual and theoretical accuracy of query results.



(a) (b)

Fig. 7. (a) Accuracy vs space trade-off for single query on TPC-H data; (b) Effect of space budget $\mathbb{B}$ on accuracy $|\delta_i - \mathsf{err}(Q_i, \epsilon_i)|$ of queries in workload for the coverings selected by the MILP.

same sets of attributes). We have arbitrarily chosen the weights and desired accuracy thresholds for each of the queries. Figure 7 (b) shows the errors, $|\delta_i - \mathsf{err}(Q_i, \epsilon_i)|$, made on each of the queries by using the coverings as identified by the MILP given in Section 6, and this for varying space budgets $\mathbb{B}$ (700000 to 130000 tuples). As expected, increasing $\mathbb{B}$ results in a better approximation of the queries. Furthermore, the increase in accuracy when increasing the budget is more noticeable for queries with high importance (*e.g.,* query Q_{19}). Solving the MILP took a few milliseconds, which is negligible in the overall query approximation process.

8 Related Work & Conclusions

Approximate query answering (AQA) in relational databases has been the subject of extensive research. We refer to the recent survey [5] for more details. Most research has focused on AQA systems that make use of concise data structures called synopses built from the database. The synopses techniques can be divided into two broad categories: non-sampling based, and sampling based. Examples of non-sampling based synopses are wavelets [6], histograms [7, 8], and kernels [9]. Sampling-based methods are the

key components of AQA systems as described in [10, 11, 1], among others. Most of these works, however, consider simple aggregate queries. A notable exception is [7] where set-valued conjunctive queries are approximated by means of a rewriting in terms of a compact histogram representation of the data. The result of this rewriting is a histogram that is an approximation of the query result. Although close in spirit to our use of coverings, [7] does not provide accuracy guarantees and cannot be easily generalised to non-histogram synopses. By contrast, our notion of covering is more general and we do provide guarantees. Furthermore, [7] considers single queries only and does not impose an upper bound on the available space. Finally, we recall that the valid covering selection problem is inspired by the sampling-based BlinkDB system [1], as mentioned in the Introduction.

▷ *Conclusions.* We have presented a formal approach for space bounded query approximation and experimentally validated it. Much more needs to be done, however: Can we enrich coverings so that they become samples that can be used to approximate aggregate queries? How to incorporate other error measures? Are there special classes of queries for which better (more compact and accurate) coverings can be computed? Can our query approximation be integrated in indexing methods or be part of the DBMS query optimiser? How can a large number of coverings be efficiently stored and accessed? Can our approach benefit from moving to other platforms, such as Apache Hive? These are just a number questions that need to be addressed.

References

1. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: BlinkDB: Queries with bounded errors and bounded response times on very large data. In: Proc. of ECCS. (2013) 29–42
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. (1979)
3. Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. Theoretical Computer Science **38** (1985) 293 – 306
4. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc. (2005)
5. Cormode, G., Garofalakis, M., Haas, P.J., Jermaine, C.: Synopses for massive data: Samples, histograms, wavelets, sketches. Found. Trends in Databases **4** (2012)
6. Chakrabarti, K., Garofalakis, M.N., Rastogi, R., Shim, K.: Approximate query processing using wavelets. In: Proc. of VLDB. (2000) 111–122
7. Ioannidis, Y.E., Poosala, V.: Histogram-based approximation of set-valued query-answers. In: Proc. of VLDB. (1999) 174–185
8. Poosala, V., Ganti, V.: Fast approximate answers to aggregate queries on a data cube. In: Proc. of SSDBM. (1999) 24–33
9. Gunopulos, D., Kollios, G., Tsotras, V.J., Domeniconi, C.: Approximating multi-dimensional aggregate range queries over real attributes. In: Proc. of SIGMOD. (2000) 463–474
10. Chaudhuri, S., Das, G., Narasayya, V.: Optimized stratified sampling for approximate query processing. ACM TODS **32**(2) (2007)
11. Gibbons, P.B., Poosala, V., Acharya, S., Bartal, Y., Matias, Y., Muthukrishnan, S., Ramaswamy, S., Suel, T.: Aqua: System and techniques for approximate query answering. Bell Labs Tech Report (1998)