Philipp Rösch, Wolfgang Lehner

Optimizing Sample Design for Approximate Query Processing

Erstveröffentlichung in / First published in:

International Journal of Knowledge-Based Organizations. 2013, 3 (4), S. 1 – 21 [Zugriff am: 05.11.2020]. IGI Global. ISSN 2155-6407

DOI: https://doi.org/10.4018/ijkbo.2013100101

Optimizing Sample Design for Approximate Query Processing

Philipp Rösch, Business Intelligence Practice, SAP Research, Dresden, Germany

Wolfgang Lehner, Database Technology Research Group, Dresden University of Technology, Dresden, Germany

ABSTRACT

The rapid increase of data volumes makes sampling a crucial component of modern data management systems. Although there is a large body of work on database sampling, the problem of automatically determine the optimal sample for a given query remained (almost) unaddressed. To tackle this problem the authors propose a sample advisor based on a novel cost model. Primarily designed for advising samples of a few queries specified by an expert, the authors additionally propose two extensions of the sample advisor. The first extension enhances the applicability by utilizing recorded workload information and taking memory bounds into account. The second extension increases the effectiveness by merging samples in case of overlapping pieces of sample advice. For both extensions, the authors present exact and heuristic solutions. Within their evaluation, the authors analyze the properties of the cost model and demonstrate the effectiveness and the efficiency of the heuristic solutions with a variety of experiments.

Keywords: *Advisor Tools, Approximate Query Processing, Computer Science, Information Systems, Random Sampling, Sampling Error Types*

INTRODUCTION

Recent studies revealed a rapid multidimensional growth in current data warehouse databases: The size of the data triples every two years and the number of queries doubles each year. Additionally, the complexity of the queries increases—a lot of queries are complex aggregation queries with joins. These queries are of explorative nature, that is, users browse through the data and search for interesting regions. (Winter, 2008)

DOI: 10.4018/ijkbo.2013100101

In order to make the data exploration reasonably applicable, short response times are essential. However, the rapid growth of data warehouse systems conflicts with the need for short response times. A common solution for this problem is the use of synopses. Synopses are concise representations that reflect the characteristics of the underlying data, and they allow for approximate query processing with significantly shorter response times. In the database literature, several kinds of synopses have been proposed like histograms (Ioannidis & Poosala, 1999; Poosala, Ioannidis, Haas, & Shekita, 1996), wavelets (Chakrabarti, Garo-

falakis, Rastogi, & Shim, 2000; Matias, Vitter, & Wang, 1998) or random samples (Vitter, 1985). From these synopses, random samples have proven to be a good choice: They provide probabilistic error bounds and they can easily be integrated into database systems. Initially, samples were mostly used for query optimization. In the last 10 years, however, the focus of database sampling has shifted more and more towards approximate query processing. Here, several techniques for online sampling have been proposed (Hellerstein, Haas, & Wang, 1997; Haas & Hellerstein, 1999; Jermaine, Dobra, Arumugam, Joshi, & Pol, 2005; Jermaine, Arumugam, Pol, & Dobra, 2007). However, the major part of research focuses on precomputed and materialized —that is, offline— sampling since online sampling has high overhead when drawing a sample and is only applicable for a small subset of queries.

In the field of offline sampling, a multitude of different sampling schemes have been proposed that are optimized for different query types, like aggregation (Chaudhuri, Das, Datar, Motwaniand, & Narasayya, 2001; Rösch, Gemulla, & Lehner, 2008), group-by (Acharya, Gibbons, & Poosala, 2000; Babcock, Chaudhuri, & Das, 2003; Rösch & Lehner, 2009) or foreign-key joins (Acharya, Gibbons, Poosala, & Ramaswamy, 1999; Gemulla, Rösch, & Lehner, 2008). While those sampling schemes provide great solutions for single (groups of) queries, the more general problem of automatic sample selection for an entire workload of a database remains almost unaddressed.

In this article, we address the problem of finding a set of samples that is to be materialized. We focus on simple random samples as those samples are easy to use and easy to maintain. Simple random samples are very general and can be used for a broad range of queries. Our solution is a sample advisor that suggests a set of samples for a set of queries specified by an expert. The sample advisor is based on a novel cost model to evaluate a sample for a given query. This cost model allows us to give advice on a sample for an individual query. To ease the usage of the sample advisor, we further propose an extension to utilize recorded workload information. In this scenario, the sample advisor selects from all the pieces of sample advices those that minimize the runtime of the workload and fit into a given memory bound. With a second extension, we are targeting the effectiveness of the sample advisor. Here, we consider the merge of samples in case of overlapping pieces of sample advice. As a result, more of the advised samples fit into the memory bound—the available memory is used more effectively—and thus, more queries can be answered very fast based on samples.

This article is an extended version of a previously published conference paper (Rösch & Lehner, 2010); we make the following (partly new) contributions:

- We analyze the database operations in the context of sample-based query processing.
- We propose a cost model for the evaluation of a sample for a given query. With this cost model, we can give a piece of sample advice for an individual query.
- Based on the cost model, we present our sample advisor in the setting of representative input queries specified by an expert.
- We propose two extensions for the sample advisor to ease the usage and to increase the effectiveness. For both extensions, we present an exact and a heuristic solution.
- With a variety of experiments, we analyze the properties of the cost model and compare the proposed algorithms. We further demonstrate the effectiveness and the efficiency of the heuristic solutions.

At the end of this article, we discuss related work and summarize the article in the conclusion. There, we also provide an outlook on future work.

DATABASE OPERATIONS AND SAMPLING

In this section, we discuss the interaction of database operations and (offline) sampling.

We specify both the prerequisites for applying the operation on a sample and the impact on the query result.

Select

Selections define specific ranges of the data that are used for further processing or returned to the user. When applied to a sample, the input of the selection is only a subset of the base data, and thus, the result of this operation may be small or even empty —especially for predicates with high selectivity. Consequently, selections lead to incomplete (intermediate) results.

Project

A projection as part of an approximate query does not differ from a projection of an exact query since it only reduces the number of attributes of the result. Hence, as long as all the required attributes are in the sample, the projection may be executed on the sample without hesitation. There is no impact on the quality of the approximate result.

Join

As for selections join operations select ranges of the processed input data based on a predicate, that is, the join condition. Unfortunately, the problem of sampling and joins is more delicate as in general a join of two samples does not result in a sample of the join of the two base tables (Chaudhuri, Motwani, & Narasayya, 1999). However, in the currently considered data warehouse scenario, tables are joined along predefined foreign-key relationships. Here, using samples of pre-computed joins or samples in the style of Join Synopses (Acharya, Gibbons, Poosala, & Ramaswamy, 1999) allow for valid samples. Hence, we can use sampling and joins; the join result on samples is incomplete.

Aggregation

Aggregations compute an aggregate from the underlying data. For a sample, the aggregate is computed on a subset of the data, and thus, the result is an estimate of the exact value. That is, the result comes with an estimation error and hence, it is inexact.

Group-By

Group-By operations segment the data into groups. In a sample, groups are represented by a subset of their elements. However, some groups—especially small ones—may be missing in the sample. Consequently, these groups are missing in the result; the result is incomplete.

Order-By

With Order-By operations, the output order of the result is specified. As for projections, there is no impact on the quality of the approximate result.

To summarize, we observe two kinds of errors for queries on samples as the approximate results are computed based on only a subset of the data: Firstly, the tuples that are not included in the sample are missing in the (intermediate) result, like for selects or joins, or may even result in completely missing groups; hence, the approximate result may be *incomplete*. Secondly, for aggregations, values of tuples included in the sample are extrapolated according to the sampling fraction while values of missing tuples do not contribute to the estimate; hence, the approximate result may be *inexact*.

A COST MODEL FOR SAMPLE EVALUATION

We now define what a sample for an individual query should look like. Finding a good sample is related to the physical design problem for indexes or materialized views. However, in the case of samples, we have to face up to a new dimension: As shown above, we additionally have to take a certain error—namely incompleteness and inexactness—into account. Aiming for a cost model, with a sample we want to achieve large decreases in the response times, and the memory cost should be low. These two goals ask for small sample sizes. At the same time, the

estimates should be close to the actual values, and the results should be preferably complete. Clearly, these goals ask for large sample sizes. Hence, there is a conflict between the goals which has to be reflected by the cost model.

As a basis, we take the cost model of the DB2 Design Advisor (Zilio, et al., 2004) (bold part) and extend it by the approximation-related parts identified in this article (non-bold part). The resulting cost model computes a weight for a sample by:

$$weight = \frac{\textbf{\textit{decrease in response time}} \cdot completeness}{\textbf{\textit{memory cost}} \cdot estimation\ \ error}$$

As can be seen, we append *completeness* (that we want to have) to the numerator and *estimation error* (that we don't like to have) to the denominator. We now analyze the individual properties in more detail. Let N be the cardinality of the base data R, with $R = \{t_1, t_2, \ldots, t_{|R|}\}$. Further, let n be the cardinality of the sample S. Now, the sampling fraction f can be expressed as $f=n/N$. Moreover, let L denote the length of a tuple in the base data, while l is the length of a tuple in the sample.

Example 1: Consider the dataset shown in Table 1. It consists of 10 orders placed by 3 different customers; for each order the prices are given. For simplification, we say that the length of each attribute is 1 for all our examples. Hence, for the example dataset, we have N=10 and L=3. Now, we want to find the optimal (simple random) sample for the following query:

```
SELECT customer, AVG(price)
FROM orders
GROUP BY customer
```

This query computes the average order price of each customer.

Decrease in Response Time

For estimating the decrease in the response time, we make the simplified assumption that both the exact and the approximate query use table scans. Indeed, this assumption often holds in practice for the complex queries focused on in this article. With this assumption the decrease in the response time Δt is proportional to N-n, and the relative decrease $\Delta t_{rel}(n)$ can be expressed as:

$$\Delta t_{rel}\left(n\right) = 1 - \frac{n}{N} = 1 - f.$$

A value of $\Delta t_{rel}(n)$=0.9 indicates that the response time can be reduced to 10% by using the sample instead of processing this query

Table 1. Orders with customer and prices

ID	CUSTOMER	PRICE
1	Smith	50
2	Smith	75
3	Jones	20
4	Jones	25
5	Jones	15
6	Brown	25
7	Brown	40
8	Brown	30
9	Brown	20
10	Brown	50

exactly, that is, we have a runtime benefit of 90%. Note that this function is independent from the dataset. It linearly decreases with increasing sample size.

Completeness

Let G be the set of groups defined by the current query. Then, $g_i \in G$, $i=1\ldots|G|$, denotes an individual group and $|g_i|$ denotes its size. Now, the probability p that at least one tuple of a group g_i is included into a sample of size n is given by:

$$p\left(g_i,n\right)=\frac{\left(N-\left|g_i\right|\right)!}{\left(N-\left|g_i\right|-n\right)!}\frac{\left(N-n\right)!}{N!}.$$

Consequently, the expected number of groups in the sample is:

$$\left|G_{sample}\left(n\right)\right|=\sum_{i=1}^{|G|}p\left(g_i,n\right),$$

and the completeness of an approximate group-by query is computed by:

$$c\left(n\right)=\frac{\left|G_{sample}\left(n\right)\right|}{\left|G\right|}.$$

The computation of the completeness for predicates (selections and joins) is much easier. Here, individual tuples are returned whose inclusion into the sample only depends on f. Hence, the completeness for predicates simply evaluates to $c(n) = n/N = f$.

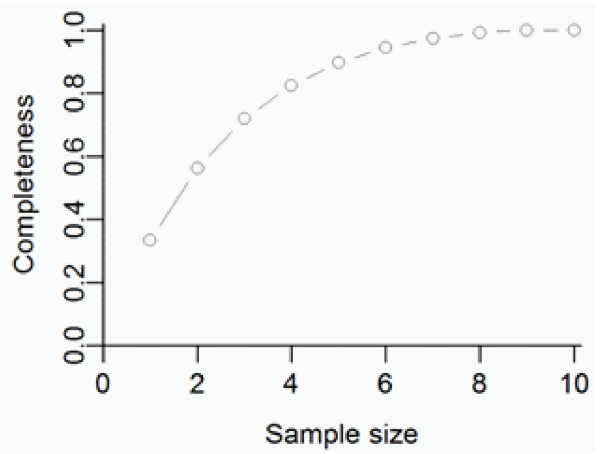
Example 2: For the example query, we have 3 groups with $|g_1=2|$, $|g_2=3|$, and $|g_3=5|$. The completeness for different sample sizes is given in Figure 1.

Memory Cost

The memory cost of a sample is made up of the number and the length of the tuples in the sample. As samples should be small, only required attributes are included into the sample. Hence, the absolute memory cost is given by $m_{abs}(n)=n\cdot l$. As for the decrease in response time, we can use the relative memory cost, which is given by

$$m_{rel}\left(n\right)=\frac{n\cdot l}{N\cdot L}.$$

Figure 1. Completeness



A value of $m_{rel}(n)=0.1$ indicates that the sample requires only 10% of the size of the base data.

Example 3: Figure 2 shows the memory cost for the example query. Here, only 2 of the 3 attributes of the base data are included into the sample.

Estimation Error

Let $a_1,\ldots,a_l$ be the attributes that are aggregated in the current query. We now show how the estimation error for these aggregates can be computed. We show this for the AVG aggregate; the computation for the SUM aggregate is similar. For COUNT aggregates, the computation has to be adapted accordingly.

Let i be the index for individual tuples of R and j be the index for attributes of R. Then, the quantity t_{ij} represents the value of attribute a_j of tuple t_i. Now, the average value of the j-th attribute is given by

$$\mu_j = \frac{1}{N}\sum_{t_i \in R} t_{ij},$$

and the standard deviation can be expressed as

$$\sigma_j = \sqrt{\frac{1}{N}\sum_{t_i \in R}\left(t_{ij}-\mu_j\right)^2}.$$

Next, let

$$RSD_j = \frac{\sigma_j}{|\mu_j|}$$

denote the relative standard deviation of attribute a_j. Note that the relative standard deviation is not defined for $\mu_j=0$ and may get very large for $\mu_j \approx 0$; thus, in our implementation, we set $RSD_j=\sigma_j$ whenever $\mu_j \in [-1,1]$. In contrast to the standard deviation, the *RSD* is unitless and can be compared across multiple attributes.

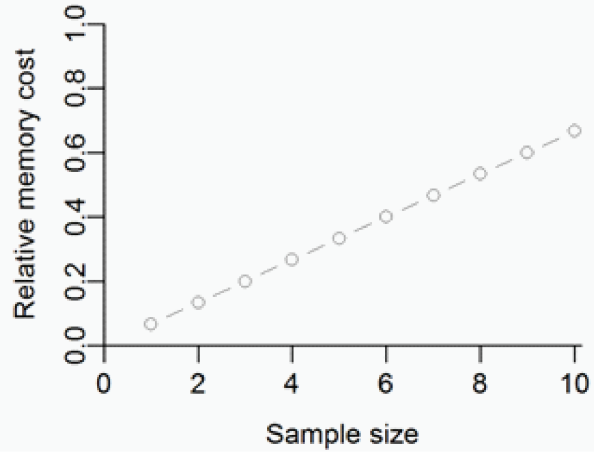
For a uniform sample of size n,

$$\hat{\mu}_j = \frac{1}{n}\sum_{t_i \in S} t_{ij}$$

is an unbiased estimate of $\mu_j(R)$. Moreover, the standard error of this estimate is:

$$\sigma_{\hat{\mu}_j}(n) = \sqrt{\frac{\sigma_j^2}{n}\left(1-\frac{n}{N}\right)} = \sigma_j\sqrt{\frac{1}{n}-\frac{1}{N}}.$$

Figure 2. Memory cost



As above, the relative standard error can be used to compare estimation errors across multiple attributes. Since $\hat{\mu}_j$ is unbiased, we can use μ_j instead:

$$RSE_{\hat{\mu}_j}(n) = \frac{\sigma_{\hat{\mu}_j}(n)}{|\mu_j|} = RSD_j \sqrt{\frac{1}{n} - \frac{1}{N}} \ .$$

Now, the overall estimation error over all the aggregation attributes is given by

$$RSE_{\hat{\mu}}(n) = \frac{1}{l} \sum_{j=1}^{l} RSE_{\hat{\mu}_j}(n) \ .$$

For queries with Group-By operations, the *RSE* is first computed for each group and then averaged over all groups.

Example 4: Given our example query, we get the following relative standard deviations: 20.0% for the first group, 20.4% for the second, and 32.6% for the third one. The resulting relative standard error for different sample sizes is shown in Figure 3.

Summing Up

We now put the pieces together. With the individual equations given above, the weight $w(n)$ of a sample of size n is computed by:

$$w(n) = \frac{\Delta t_{rel}(n) \cdot c(n)}{m_{rel}(n) \cdot RSD(n)} \ .$$

Example 5: For the example query, Figure 4 shows the weight of the sample for different sample sizes with a maximum for n=2. This curve emphasizes that the sample size should be chosen with care: Small samples have low weights due to large estimation errors and many missing tuples, while large samples suffer from low runtime benefits and high memory costs. □

A deeper look at the weight computation reveals two shortcomings: First, the estimation error only influences the amplitude of the curve but not its position and second, the maxima of different samples differ both in amplitude and position, which makes comparisons of weights impractical. As a solution, we propose the following normalization:

$$\overline{w}(n) = \frac{w(a \cdot n)}{a}$$

Figure 3. Relative standard error

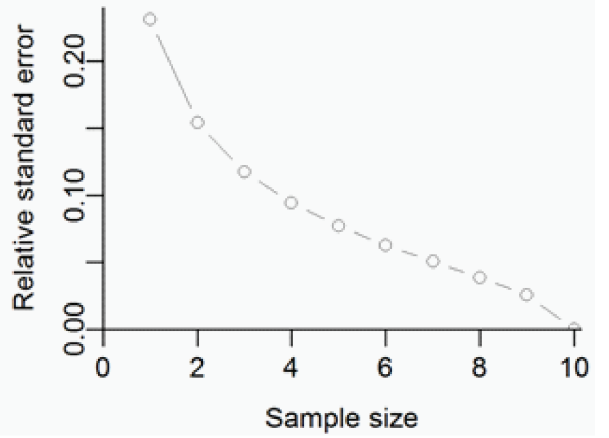
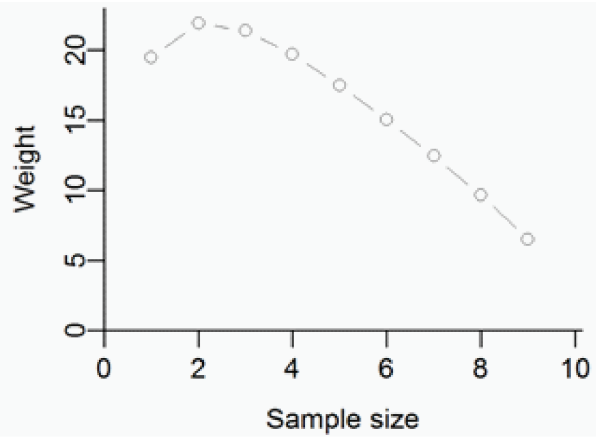


Figure 4. Weight for different sample sizes



with a=max($w(n)$). This normalization is based on the observation that high deviations of the data—and thus, large estimation errors—result in low weights. However, in order to provide good estimates, we need large samples for data with high deviations.

Example 6: Consider a dataset consisting of N=100 tuples and 10 groups with 10 tuples each. Figure 5 shows the sample weights for varying relative standard deviations of the base data without normalization. As can be seen, this variation only influences the amplitude, and larger RSDs result in lower weights. The normalized weights are given in Figure 6. Here, larger relative standard deviations result in larger optimal sample sizes.

A SAMPLE ADVISOR FOR APPROXIMATE QUERY PROCESSING

Based on the proposed cost model, we next introduce our sample advisor. The goal of this advisor is to recommend samples for some given queries. Those queries could either be some carefully chosen queries representing

Figure 5. Impact of the relative standard deviation

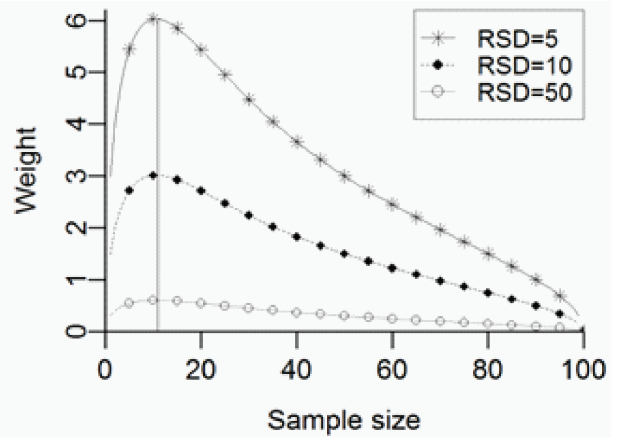
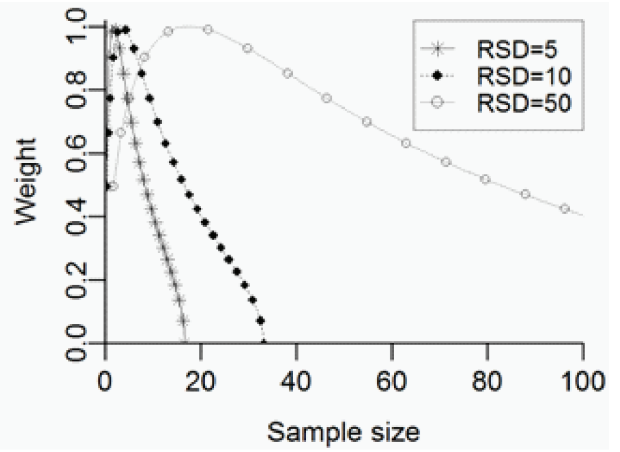


Figure 6. Impact of the relative standard deviation (normalized)



the expected workload or it could be recorded workload information. We start with addressing the first case and show how to also support the second case with our first extension of the sample advisor.

As mentioned, we primarily are targeting an expertise-based sample configuration. In this setting, an expert identifies the (few) most expensive analytical queries and provides them to the sample advisor; clearly, the number of queries depends on the memory that is available for the samples. Now, our sample advisor computes for each of the given queries a piece of sample advice. Such a piece of sample advice SA comprises the base data R, the attributes A to be included into the sample and the sample size n, thus SA=(R,A,n). The first two values can easily be derived from the query; the last one is determined with the cost model given above. Hence, in order to compute the piece of sample advice for a given query q, we first set R to the base data of q and A to the set of attributes referenced by q. Then, we determine the sample size with the following two steps:

1. Scan the base data of the query once and compute for each group the relative standard deviation and the size.
2. Iterate over different sample sizes and compute the weight. Remember the sample size with the largest weight.

These two steps are repeated for all of the given queries, and finally, the optimal expertise-based sample configuration SC_E, i.e., the result of the sample advisor, consist of all the pieces of sample advice SA_i for the expert-given queries q_i.

Considering the effort of the sample advisor, we see that the first step only depends on the cardinality of the base data, and thus, its effort is fixed. The effort of the second step, however, depends on the number of regarded sample sizes. Here, some considerations may reduce the effort: First, the weight function has a single maximum. This maximum can efficiently be found by algorithms like hill climbing or binary search. Second, we can optionally define upper and lower bounds for the sample sizes since, for example, samples smaller than 0.1% are expected to provide very imprecise results or samples larger than 10% are considered as too expensive and the benefit is low. In order to further reduce the effort of the weight computation itself, we can use the completeness for sampling *with replacement* as an approximation (and lower bound) for the completeness of sampling without replacement:

$$\tilde{c} = \frac{1}{|G|} \sum_{i=1}^{|G|} \left(1 - \left(1 - \frac{|g_i|}{n} \right)^n \right) .$$

With $\tilde{c}$, no expensive factorials are required. In this equation, the innermost part is the probability of the event that with n draws no tuple of group g_i is chosen. Now, the complementary probabilities (at least one tuple of g_i is drawn) sum up to the expected number of groups in the sample, and we get the completeness be dividing the expected number of groups in the sample by the number of groups in the base data.

Extension I: Workload-based Sample Configuration

Besides relying on an expert one common approach is to utilize recorded workload information. Clearly, in such a case we cannot materialize the samples for all the queries. We thus propose the following extension of our sample advisor. Note that this extension does not result in an optimal solution but shows how the approach proposed above can easily be adapted to workload-based scenarios.

Let the workload W be a (multi-) set of queries with $W=\{q_1,\ldots,q_k\}$. Before we compute the workload-based sample configuration SC_W, we preprocess the workload by eliminating all but aggregation queries so that the workload only consists of queries relevant for approximate query processing. Further, the multiset of queries is transformed to a set by replacing the queries by (query, counter) pairs and merging duplicate queries. During this merge, predicates of the queries are ignored. The reason for this is that samples should be very general; otherwise, the samples would be similar to materialized views and we would severely restrict their applicability. After the merge, we get $W_{AQP} = \{(q_1,c_1),\ldots,(q_l,c_l)\}$.

With a memory constraint M, we get the following two steps:

1. Compute the optimal sample for each query of W_{AQP}. The result is a candidate set with pieces of sample advice $C = \{SA_1,\ldots,SA_k\}$.
2. Compute the optimal sample configuration SC_W of size M based on the candidate set C from the first step.

Obviously, with the first step we utilize our weight function and the computation of the expertise-based sample configuration. For the second step, we need a measure to compare different configurations. As a desirable sample configuration is characterized by a preferably low overall runtime of the workload, we use this overall runtime as our measure and try to minimize it. As before, we assume table scans, and since we are not interested in actual response times, we simply use $r=n\cdot l$ as the response time for approximate queries and $r=N\cdot L$ for all queries with no sample in SC_W as they have to be answered with the base data. Now, the measure $\mathcal{F}$ of a sample configuration is:

$$\mathcal{F}\left(SC\right) = \sum_{q_i \in W_{AQP}} r_i \ .$$

The goal of the second step is to find the sample configuration that fits into M and minimizes $\mathcal{F}$. Obviously, the exact solution is an instance of the knapsack problem, which is known to be NP-hard.

Optimal Solution: To find the optimal solution (based on C), we consider all subsets C' of the candidate set C that fit into M and compute $\mathcal{F}$. The final sample configuration is the subset with the minimal measure $\mathcal{F}$.

Greedy Solution: The basic idea of the greedy approach is to successively add the most valuable candidates to the sample configuration until the memory bound is hit. We first order the candidate set C by the following score in descending order:

$$score\left(SA_i\right) = c_i \cdot \frac{N_i \cdot L_i}{n_i \cdot l_i} \ .$$

This score is composed of the query counter to account for the frequency of the sample usage—the more often a sample is used the more beneficial gets its materialization—and the inverse of the relative memory cost. The second part of the score makes samples with smaller optimal sampling fractions to be considered

more valuable. This is motivated by the fact that those samples have low storage requirements and offer large response time benefits.

Next, we start with an empty sample configuration SC_W and successively add the candidates to SC_W in the given order until the next candidate does not fit into M. At this point, we allow to skip individual candidates in order to add those that still fit into M even if their score is lower. Once the memory bound is hit or none of remaining candidate is small enough to still fit into M we have our final sample configuration SC_W.

Example 7: Consider 2 relations R_1 and R_2 with N_1=100 and N_2=30 as well as L_1=5 and L_2=3. With the candidates given in Table 2, let $C = C_{ordered} = \{SA_1, SA_2, SA_3, SA_4\}$ be the (already ordered) candidate set. This table also shows the memory consumption—as defined by $n \cdot l$—and the scores of the candidates given that c_i=1 for all $q_i \in W_{AQP}$. Now, let M=40. We successively add the elements of C to the initially empty sample configuration SC_W. After having added SA_2, the memory consumption of SC_W adds up to 23. Now, SA_3 does not fit into M while SA_4 does. Consequently, we skip SA_3 and add SA_4. The final sample configuration is $SC_W = \{SA_1, SA_2, SA_4\}$. The measure $\mathcal{F}$ of this sample configuration is

$$\mathcal{F}\left(SC_W\right) = 8 + 15 + 500 + 4 = 527.$$

Extension II: Merging Pieces of Sample Advice

Besides the 'simple' selection of samples, we propose a second extension that additionally considers the possibility of merging multiple pieces of sample advice. This idea is based on the following observation: In typical OLAP scenarios, many queries have the same base data—especially if the predicates are disregarded—and the referenced attributes often overlap. Hence, up to now there are samples in SC_W with the same base data and overlapping attribute sets. In order to use the available memory more effectively, we consider to merge those samples. Then, the queries of both samples can be answered by the single merged sample; the redundancy in SC_W decreases.

Prerequisites for merging two pieces of sample advice SA_i and SA_j are the same base data $R_i = R_j$ as well as overlapping attributes, i.e., $A_i \cap A_j \neq \varnothing$. The merged piece of sample advice $SA_{i+j} = (R_{i+j}, A_{i+j}, n_{i+j})$ is computed by:

- $R_{i+j} = R_i = R_j$,
- $A_{i+j} = A_i \cup A_j$, and
- $n_{i+j} = \max(n_i, n_j)$.

When merging two pieces of sample advice, we take the maximum of the sample sizes for the following two reasons: First, decreasing n results in (considerably) higher errors (estimation error and missing tuples) and second, increasing n has less impact on $\bar{w}$ than decreasing it.

However, aside from the prerequisites, one has to verify whether or not a merge is beneficial. Clearly, a query q_i of SA_i must read $A_{i+j} \setminus A_i$ additional attributes and $n_{i+j} - n_i$ additional tuples when using the sample of SA_{i+j} instead of the sample of SA_i. Having $\mathcal{F}$ in mind, a merge is only beneficial if the overall runtime of the workload decreases, and thus, if the merge frees enough memory to add an additional sample to SC_W.

Table 2. Sample advisor candidates

Piece of Sample Advice	Base Data	Attributes	Sample Size	Memory	Score
SA_1	R_1	$\{A_1, A_2\}$	4	8	62.5
SA_2	R_1	$\{A_3, A_4, A_5\}$	5	15	33.3
SA_3	R_1	$\{A_2, A_3, A_4\}$	7	21	23.8
SA_4	R_2	$\{A_1, A_3\}$	2	4	22.5

Optimal Solution: For the optimal solution, we consider all possible merges. For each considered merge, we replace the respective pieces of sample advice by the merged piece of sample advice and proceed as in the strategy without merge. Obviously, this procedure is very expensive.

Greedy Solution: With the greedy approach for this extension, we initially proceed as in the greedy approach without merging given above: We order the candidate set C by the *score* value and start by adding the pieces of sample advice into an initially empty sample configuration SC_W. However, when reaching a candidate that does not fit into the memory bound M, we now try to greedily merge individual pieces of sample advice. Clearly, the goal is to free enough memory to add the current piece of sample advice SA_i to the sample configuration SC_W. Therefore, we consider all possible merges of the sample advice currently in $\mathrm{SC_W} \cup \mathrm{SA_i}$. From these merges, we choose the most beneficial one, i.e., the merge that frees the most memory. In the case of equal memory consumptions, we additionally consider the overall runtime $\mathcal{F}(\mathrm{CS})$ and choose the one with the minimal $\mathcal{F}(\mathrm{CS})$. If the available memory is still too low, we again look for the most beneficial merge, but this time, we replace the two pieces of sample advice chosen to merge with the merged piece of sample advice. We repeat this procedure until either enough memory is freed or no more beneficial merges are possible. In the former case, we perform the merges, in the latter case, we skip the current candidate and proceed with the next one until all candidates are considered or the memory bound is hit. Note, in the greedy merge process, the repeated procedure of finding the most beneficial merge can be done very efficiently —all we need are the sample sizes, the tuple lengths and the overlapping attributes of the merge candidates.

Example 8: Consider again the setting of Example 7. We start by adding SA_1 and SA_2 to the initially empty sample configuration SC_W. Next, SA_3 is considered. SA_3 does not fit into M and thus, we try to find a merge. Since the attribute sets of SA_1 and SA_2 are disjoint, there is no overlap and we do not merge these pieces of sample advice. SA_3, however, meets the condition to be merged with either SA_1 or SA_2 and we determine the more beneficial merge. For both SA_{1+3} and SA_{2+3}, we have m_{abs}=28 and thus, we also have to consider the overall runtime. With SA_{1+3}, the overall runtime of q_1, q_2, and q_3 sums up to r=28+15+28=71, while with SA_{2+3}, the overall runtime is only r=8+28+28=64. Thus, we prefer SA_{2+3} and we have $SC_W=\{SA_1, SA_{2+3}\}$ with a memory consumption of 36. In the next step, we add SA_4 to SC_W, and our final sample configuration is $\mathrm{SC_W}=\{\mathrm{SA_1}, \mathrm{SA_{2+3}}, \mathrm{SA_4}\}$. For this sample configuration, we get

$$\mathcal{F}(\mathrm{SC}) = 8 + 28 + 28 + 4 = 68\,,$$

which is significantly lower —and thus, better— than $\mathcal{F}(\mathrm{SC}) = 527$ of the greedy approach without merging pieces of sample advice.

EXPERIMENTS

We ran a variety of experiments in order to analyze the cost model and to compare the strategies for the construction of workload-based sample configurations. With the evaluation of the cost model, we implicitly evaluate the properties of the expertise-based sample advisor. For the workload-based sample configurations, we compared the strategy without merging samples (*NoMerge*) with the strategy that considers the merge of samples (*Merge*). We further evaluated the effectiveness and the efficiency of the heuristic algorithms, i.e., *NoMergeGreedy* and *MergeGreedy*. We experimented with well-defined synthetic datasets in order to discover the impact of certain "data formations" like the number of groups, the group sizes or the variance of the data on the weight function and the resulting sample configuration. Finally, we ran experiments on a large real-world dataset consisting of retail data to also show the practical applicability.

Note that the considered algorithms are deterministic with respect to the resulting sample configuration. Hence, our measure $\mathcal{F}$ for the comparison of the proposed algorithms can be computed analytically.

A short summary of our results is as follows:

- The cost model reflects the characteristics of the underlying data like the group sizes and their distribution. Hence, we are able to give an appropriate piece of sample advice for a given query.
- The *Merge* strategy results in considerably lower runtimes of the workload than the *NoMerge* strategy and thus, considering merges of samples is highly beneficial.
- The heuristic algorithms significantly reduce the computation costs and provide similar results like the exact algorithms.

Experimental Setup

We implemented the sample advisor on top of DB2 using Java 1.6. The experiments were conducted on an Athlon AMD XP 3000+ system running Linux with 2GB of main memory.

Cost Model: For the evaluation of our cost model, we generated a small synthetic dataset R with N=1000 tuples and L=10 attributes. The specific properties of this dataset are given in Table 3. Unless stated otherwise, the parameters take the value given in the last column ('Default value').

Sample Configuration: For the evaluation of the sample configurations, we used two different datasets:

- A very small synthetic dataset with N=100 tuples and L=15 attributes. For this dataset, we used a workload of 5 carefully chosen queries.
- A large real-world dataset of retail data. The fact table of this dataset consists of 13,223,779 tuples with L=15 attributes, of which 5 attributes are used for grouping and 8 attributes are used for aggregation. Additionally, we chose 2 of the dimension tables which also consist of a few aggregation attributes. The workload of this dataset consists of 15 typical OLAP queries.

Analysis of the Cost Model

In the first part of our experiments, we analyzed the proposed cost model. Therefore, we varied several parameters of the base data and computed the weight for samples of different sizes, each with l=4 attributes. We varied the number of groups, the skew of the group sizes, and the relative standard deviation of the aggregation values as shown in the next.

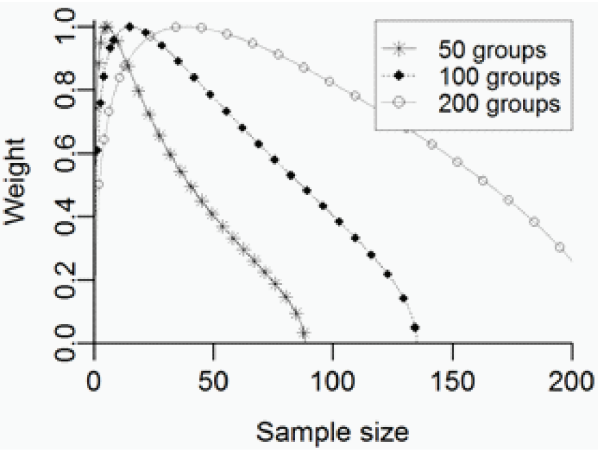
Number of Groups: In the first experiment, we varied the number of groups from 50 to 200 groups and computed the weight for different sample sizes. As shown in Figure 7, the number of groups influences the optimal sample size: For 50 groups, the optimal sample size is 5 tuples, while for 200 groups the optimal sample size is 38 tuples. The reason is that the more groups we have the smaller they are. Smaller groups, in turn, are more likely to be missing in a sample and thus, the optimal sample size increases with increasing number of groups.

Skew of Group Sizes: Next, we varied the skew of the group sizes. We chose a Zipfian distribution with z values ranging from z=0 (uniform) to z=1.4 (highly skewed). Here, the value of z=0.86 results in a typical 90-10 distribution. The result is shown in Figure 8. As can be seen, larger skew results in larger optimal sample sizes. Again, the reason is that smaller groups are more likely to be missing: The more

Table 3. Parameters for the experiments

Parameter	Range of Values	Default Value
Number of groups	50-200	100
Skew of group sizes	0-1.4	0.86
Average RSD	5-50	15

Figure 7. Number of groups



skewed the group sizes the more small groups are in the base data. Hence, the larger the skew the larger the optimal sample size.

Relative Standard Deviation of Aggregation Values: Finally, we varied the relative standard deviation of the base data. Inspired by our real-world dataset, we chose values from *RSD*=5 to *RSD*=50. As the RSD directly influences the estimation error (see the discussion of the estimation error above), larger RSDs result in larger estimation errors and thus, in larger optimal sample sizes. This is also shown in Figure 9.

Sample Configuration

In the second part of our experiments, we compared the strategies and the algorithms for the computation of the sample configuration.

Synthetic Dataset: As stated above, we first evaluated our algorithms on a small dataset and we carefully selected 5 queries. With our cost model, we got the 5 pieces of sample advice illustrated in Figure 10 with the scores.

Next, we computed the optimal sample configuration with all four algorithms: *NoMerge*, *NoMergeGreedy*, *Merge*, and

Figure 8. Skew of group sizes

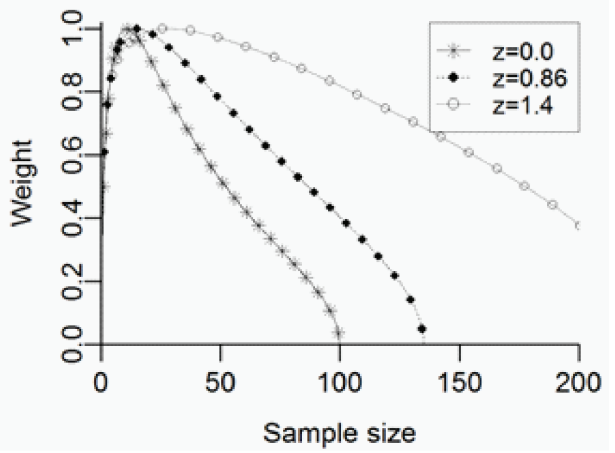
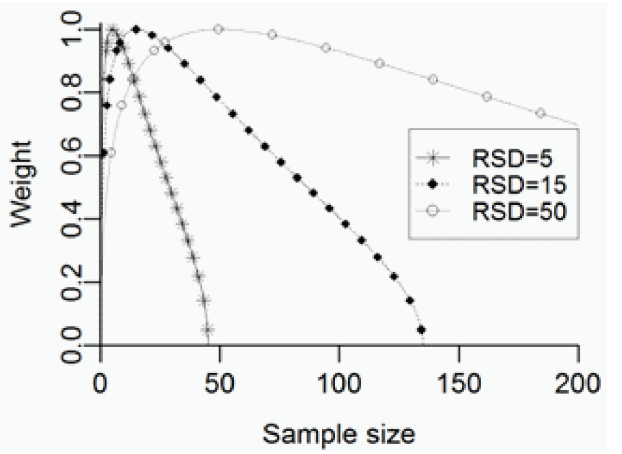


Figure 9. Relative standard deviation



MergeGreedy. We varied the memory bound from M=0 to M=79 attributes (for M=79, all samples fit into the memory bound) and computed $\mathcal{F}$, i.e., the runtime of the 5 queries in terms of number of attributes to read. As can be seen in Figure 11, there are some memory bounds where the merge of samples considerably decreases $\mathcal{F}$, e.g., for M=53, the merge decreases $\mathcal{F}$ from 456,200 to 185,515 by a factor of about 2.5. The impact of the strategy and the greedy proceeding can be seen in the close-up on $\mathcal{F}$ for M=20 to M=45, see Figure 12: For M=25, *Merge* and *MergeGreedy* merge the pieces of sample advice SA_2 and SA_3, while the sample configurations of *NoMerge* and *NoMergeGreedy* only consist of SA_2. For M=28, *MergeGreedy* selects SA_1 and thus, even performs worse than for M=27. This is a drawback of the greedy proceeding. Furthermore, for M=30 *NoMerge* selects SA_2 and SA_3 and thus, results in a better sample configuration than *MergeGreedy* that still selects SA_1. These results

Figure 10. Pieces of sample advice for the synthetic dataset

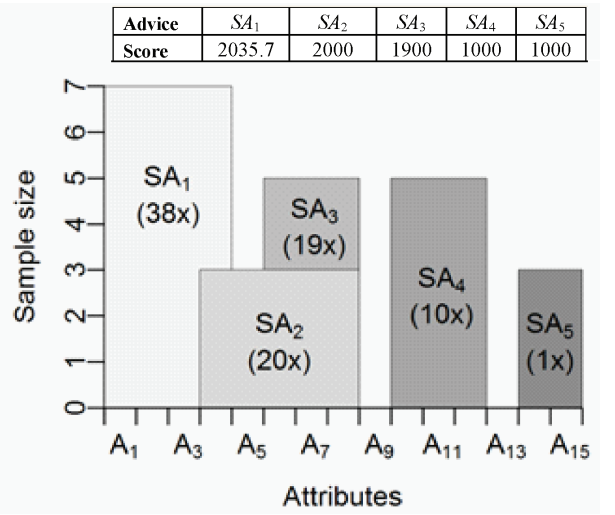
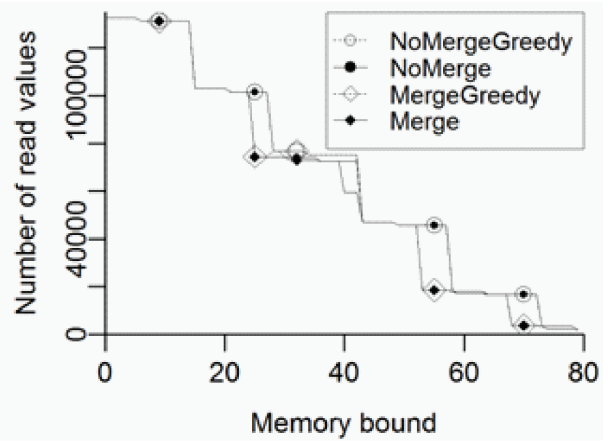


Figure 11. Performance (in terms of $\mathcal{F}$) of the sample configurations

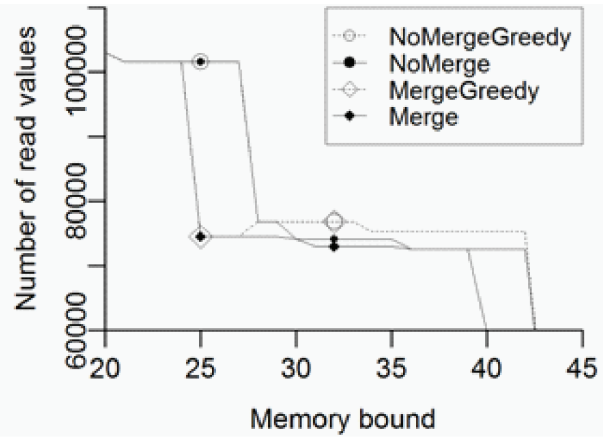


show that for our carefully chosen queries, the effectiveness of *MergeGreedy* (temporarily) may decrease for increasing memory bounds, and that *NoMerge* may be more effective than *MergeGreedy*. All in all, *Merge* always results in the best configuration, while *NoMergeGreedy* always results in the worst.

At this point, one might ask why not to merge all pieces of sample advice and use the resulting single sample for the approximate query answering. The reason for considering multiple samples for a single table is the high overhead induced by the merge: The sample size of the merged sample is the maximum sample size of all pieces of advice, and it contains all referenced attributes. This, in turn, results in high minimal memory requirements. For our synthetic dataset, the memory bound M must be at least M=7·13=91. In contrast, with our approach all the samples can be materialized for M=79 without overhead even without considering any merge of pieces of sample advice. The large overhead of the single sample also directly increases the response times of all queries and thus, $\mathcal{F}$ gets large.

Real-World Dataset: Our next experiments were conducted on our large real-world dataset. The effectiveness of the different algorithms

Figure 12. Performance (in terms of F) of the sample configurations, detailed view



is given in Figure 13 and 14. Note that in order to make the results easier to interpret, we used relative memory bounds in the plots. Again, we computed $\mathcal{F}$ for different memory bounds. The results depicted in Figure 13 show that it is beneficial to merge pieces of sample advice. Further, they clearly demonstrate the effectiveness of the greedy algorithms. To make the benefit of merging pieces of sample advice even clearer, Figure 14 depicts the improvement achieved by considering those merges. As can be seen, the improvement quickly reaches 100%, i.e., the number of attributes that have to be read halves due to the merges. For specific memory bounds, the improvement exceeds 300% for the optimal solutions and 500% for the greedy solutions. All in all, for a broad range of memory bounds it is very beneficial to consider merges of pieces of sample advice.

Runtimes: In a final experiment, we compared the efficiency of our algorithms. We varied the number of candidates from $|C|$=1 to $|C|$=15 and computed the sample configuration with all of our algorithms. Figure 15 illustrates the time to compute the sample configurations. These values are averaged over 50 runs, and we used a relative memory bound of 6%. The plot clearly shows the benefit of the greedy proceeding: While the effort of the optimal solutions quickly gets high and exponentially increases with the number of candidates, the effort for the greedy solutions is significantly lower. For *NoMergeGreedy*, the effort is logarithmic due to the ordering of the candidates, and for *MergeGreedy*, the effort is quadratic due to the greedy merge. In an additional run, we measured the times for the greedy algorithms for $|C|$=75 candidates. Here, the computation of the sample configuration still took less than a millisecond for *NoMergeGreedy* and about 1.3 seconds for *MergeGreedy*.

To summarize, our results on synthetic and real-world datasets show that the merge of pieces of sample advice is beneficial. It significantly reduces the runtime of the given workload. Additionally, the greedy algorithms are very efficient and effective – they often result in the same sample configurations as the exhaustive approaches while requiring only a fraction of the time to compute the configuration.

RELATED WORK

In this section, we review related work in the field of automatic sample selection for the approximate answering of queries in databases. This review reveals that the problem of finding an optimal sample configuration is barely studied. Some initial ideas in this field are

Figure 13. Performance (in terms of $\mathcal{F}$) of the sample configurations for the real-world dataset

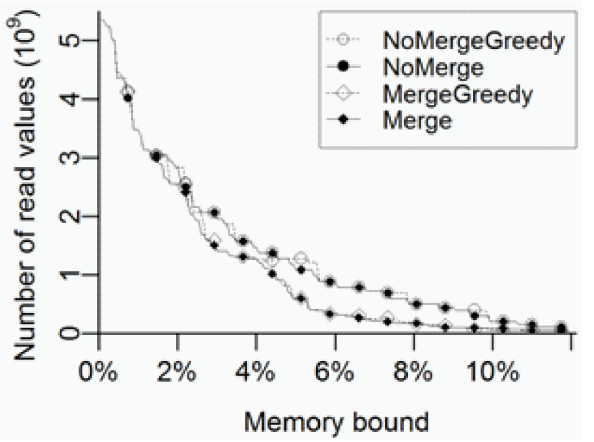
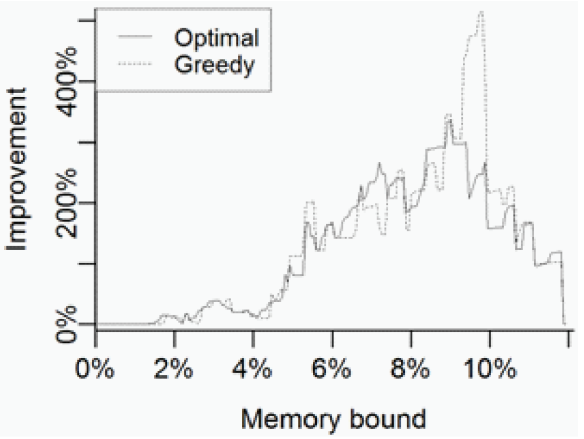


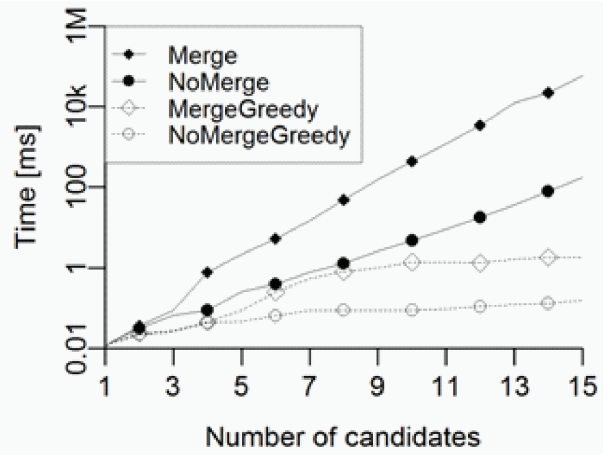
Figure 14. Performance improvement (in terms of $\mathcal{F}$) achieved by merging pieces of sample advice



those of (Chaudhuri & Narasayya, 2001) and (König & Weikum, 2002). For (Chaudhuri & Narasayya, 2001) the strategy is to select from all possible synopses those that influence the query plan or the execution costs. However, the problem of having a memory bound and hence, the partition of the available space is not regarded. On the other hand, the solution in (König & Weikum, 2002) proposes a technique for both the selection of synopses and for the partitioning of the available memory. However, all these considerations build on spline-based synopses, so that the solutions cannot easily be used for the selection of samples. Moreover, the focus of both solutions is the selectivity estimation where the approximation error bears another meaning as it is not directly passed to the user. To the best of our knowledge, there is no solution for the determination of an optimal, memory-bounded set of samples for a given set of queries.

Besides the field of automatic sample selection, the problem is related to the physical design problem. Here, additional structures like

Figure 15. Runtimes of the algorithms (log scale)



indexes (Finkelstein, Schkolnick, & Tiberio, 1988; Chaudhuri & Narasayya, 1997) and materialized views (Gupta & Mumick, 2005; Zilio, et al., 2004) as well as the combination of indexes and materialized views (Agrawal, Chaudhuri, & Narasayya, 2000) are proposed. Most of these solutions use a what-if interface (Chaudhuri & Narasayya, 1998) and ask the optimizer for the benefit. However, optimizer calls are expensive and estimated costs may be far off, especially when attributes have correlated data distributions (Gebaly & Aboulnaga, 2008). Moreover, the extension of the what-if interface in order to estimate both the cost and the error introduced by approximate query processing might be a complex task. The alternative solution is to define an explicit cost model as done by the approaches in (Gupta & Mumick, 2005), (Zilio, et al., 2004) or (Grund, et al., 2010). Those cost models constitute a good starting point; our weight function was inspired by that of (Zilio, et al., 2004). The existing cost models, however, cannot directly be used for a sample advisor as they are not designed for the context of approximate query processing, e.g., they do not account for estimation errors or incompleteness.

After this review of related work, we conclude the article with the following summary.

CONCLUSION

In this article, we proposed a sample advisor for the approximate answering of analytical queries. This sample advisor is based on a novel cost model for the sample selection. We proposed a weight function that enables us to give a piece of sample advice for any individual query. Building on that, we have shown how to compute an expertise-based sample configuration for individually specified queries. Additionally, we considered two extensions of the sample advisor. In the first extension, the sample advisor uses recorded workload information as input to compute a —now memory-bounded— sample configuration. Here, the sample advisor selects from the available pieces of sample advice those that minimize the runtime of the given workload. The second extension provides a more sophisticated solution by considering the merge of pieces of sample advice, which may significantly reduce the overall runtime of the given workload. For both extensions, we presented and evaluated an exact and a heuristic algorithm. Our experiments have shown that the merge of samples is almost always beneficial and provides large runtime savings for the given workload. Furthermore, our greedy algorithms significantly reduce the computation cost with only low impact on the effectiveness.

Our next steps include the consideration of more sophisticated sampling schemes. Those sampling schemes may significantly reduce the estimation error. However, computing—or at least estimating—these estimation errors without drawing the sample is often considerably more complex than for simple random samples. Moreover, the specialization of those sampling schemes also makes the computation of the sample configuration more complex. Here, novel selection and merge strategies have to be developed in that context.

The problem of the sample selection for an incoming query has some similarities with the first part of the sample configuration computation. As this problem is also unresolved in many systems, our results —especially the cost model— may also be reused as a starting point for novel solutions.

REFERENCES

Acharya, S., Gibbons, P. B., & Poosala, V. (2000). *Congressional Samples for Approximate Answering of Group-By Queries* (pp. 487–498). ACM SIGMOD Intl. Conf. on Management of Data. doi:10.1145/342009.335450

Acharya, S., Gibbons, P. B., Poosala, V., & Ramaswamy, S. (1999). *Join Synopses for Approximate Query Answering* (pp. 275–286). ACM SIGMOD Intl. Conf. on Management of Data.

Agrawal, S., Chaudhuri, S., & Narasayya, V. R. (2000). *Automated Selection of Materialized Views and Indexes in SQL Databases* (pp. 496–505). Intl. Conf. on Very Large Data Bases.

Babcock, B., Chaudhuri, S., & Das, G. (2003). *Dynamic Sample Selection for Approximate Query Processing* (pp. 539–550). ACM SIGMOD Intl. Conf. on Management of Data. doi:10.1145/872819.872822

Chakrabarti, K., Garofalakis, M. N., Rastogi, R., & Shim, K. (2000). *Approximate Query Processing Using Wavelets* (pp. 111–122). Intl. Conf. on Very Large Data Bases.

Chaudhuri, S., Das, G., Datar, M., Motwaniand, R., & Narasayya, V. R. (2001). *Overcoming Limitations of Sampling for Aggregation Queries* (pp. 534–544). Intl. Conf. on Data Engineering. doi:10.1109/ICDE.2001.914867

Chaudhuri, S., Motwani, R., & Narasayya, V. (1999). *On Random Sampling over Joins* (pp. 263–274). ACM SIGMOD Intl. Conf. on Management of Data.

Chaudhuri, S., & Narasayya, V. R. (1997). *An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server* (pp. 146–155). Intl. Conf. on Very Large Data Bases.

Chaudhuri, S., & Narasayya, V. R. (1998). *AutoAdmin "what-if" Index Analysis Utility* (pp. 367–378). ACM SIGMOD Intl. Conf. on Management of Data.

Chaudhuri, S., & Narasayya, V. R. (2001). Automating Statistics Management for Query Optimizers. *IEEE Transactions on Knowledge and Data Engineering*, *13*(1), 7–20. doi:10.1109/69.908978

Finkelstein, S., Schkolnick, M., & Tiberio, P. (1988). Physical Database Design for Relational Databases. *ACM Transactions on Database Systems*, *13*(1), 91–128. doi:10.1145/42201.42205

Gebaly, K. E., & Aboulnaga, A. (2008). *Robustness in Automatic Physical Database Design* (pp. 145–156). Intl. Conf. on Extending Database Technology.

Gemulla, R., Rösch, P., & Lehner, W. (2008). *Linked Bernoulli Synopses: Sampling Along Foreign-Keys* (pp. 6–23). Intl. Conf. on Statistical and Scientific Database Management.

Grund, M., Krüger, J., Plattner, H., Zeier, A., Cudre-Mauroux, P., & Madden, S. (2010). HYRISE - A Main Memory Hybrid Storage Engine}. *PVLDB*, *4*(2), 105–116.

Gupta, H., & Mumick, I. S. (2005). Selection of Views to Materialize in a Data Warehouse. *IEEE Transactions on Knowledge and Data Engineering*, *17*(1), 24–43. doi:10.1109/TKDE.2005.16

Haas, P. J., & Hellerstein, J. M. (1999). *Ripple Joins for Online Aggregation* (pp. 287–298). ACM SIGMOD Intl. Conf. on Management of Data.

Hellerstein, J. M., Haas, P. J., & Wang, H. J. (1997). *Online Aggregation* (pp. 171–182). ACM SIGMOD Intl. Conf. on Management of Data.

Ioannidis, Y. E., & Poosala, V. (1999). *Histogram-Based Approximation of Set-Valued Query-Answers* (pp. 174–185). Intl. Conf. on Very Large Data Bases.

Jermaine, C., Arumugam, S., Pol, A., & Dobra, A. (2007). *Scalable Approximate Query Processing with the DBO Engine* (pp. 725–736). ACM SIGMOD Intl. Conf. on Management of Data. doi:10.1145/1247480.1247560

Jermaine, C., Dobra, A., Arumugam, S., Joshi, S., & Pol, A. (2005). *A Disk-Based Join with Probabilistic Guarantees* (pp. 563–574). ACM SIGMOD Intl. Conf. on Management of Data.

König, A. C., & Weikum, G. (2002). *A Framework for the Physical Design Problem for Data Synopses* (pp. 627–645). Intl. Conf. on Extending Database Technology. doi:10.1007/3-540-45876-X_39

Matias, Y., Vitter, J. S., & Wang, M. (1998). *Wavelet-Based Histograms for Selectivity Estimation* (pp. 448–459). ACM SIGMOD Intl. Conf. on Management of Data.

Poosala, V., Ioannidis, Y. E., Haas, P. J., & Shekita, E. J. (1996). *Improved Histograms for Selectivity Estimation of Range Predicates* (pp. 294–305). ACM SIGMOD Intl. Conf. on Management of Data. doi:10.1145/233269.233342

Rösch, P., Gemulla, R., & Lehner, W. (2008). *Designing Random Sample Synopses with Outliers* (pp. 1400–1402). Intl. Conf. on Data Engineering.

Rösch, P., & Lehner, W. (2009). *Sample Synopses for Approximate Answering of Group-By Queries* (pp. 403–414). Intl. Conf. on Extending Database Technology. doi:10.1145/1516360.1516408

Rösch, P., & Lehner, W. (2010). A Sample Advisor for Approximate Query Processing. *East European Conf. on Advances in Databases and Information Systems*, (pp. 490-504).

Vitter, S. (1985). Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 37–57. doi:10.1145/3147.3165

Winter, R. (2008). Scaling the Data Warehouse. *Intelligent Enterprise*.

Zilio, D. C., Zuzarte, C., Lohman, G. M., Pirahesh, H., Gryz, J., Alton, E., & Valentin, G. (2004). *Recommending Materialized Views and Indexes with IBM DB2 Design Advisor* (pp. 180–188). Intl. Conf. on Autonomic Computing.

Philipp Rösch is Senior Researcher in the Business Intelligence practice at SAP Research. He joined SAP Research in 2009 after finishing his PhD at the Database Technology Research Group of Professor Wolfgang Lehner at the Dresden University of Technology, Dresden. There, his research focus was the approximate answering of queries based on random samples for the analysis of large-scale datasets. Now, he is leading and contributing to different topics in the field of hybrid-store databases as well as realtime and predictive analytics. Recently, also mobile and extremely simplified user interaction got into his focus.

Wolfgang Lehner is Full Professor and head of the database research group at the Dresden University of Technology, Dresden. He holds a diploma in Computer Science from University Erlangen-Nuremberg. He earned his PhD in Computer Science and his habilitation also from University of Erlangen-Nuremberg. He was visiting scientist at multiple renowned research institutions like IBM Almaden, SAP Labs, Microsoft Research. He has published more than 150 papers and multiple text books. His current research interests comprise support for Realtime/ Righttime-Analytics in Data-Warehouse infrastructures, main-memory database technology for analytical and transactional workloads, and support of advanced analytics (Mining, Forecasting, etc.) in database systems.