Sample Trace: Deriving Fast Approximation for Repetitive Queries

Feng Yu
Department of Computer Science
and Information System
Youngstown State University
Email: fyu@ysu.edu

Wen-Chi Hou
Department of Computer Science
Southern Illinois University Carbondale
Email: hou@cs.siu.edu

Cheng Luo
Department of Mathematics and
Computer Science
Coppin State University
Email: cluo@coppin.edu

***Abstract*—Repetitive queries refer to those queries that are likely to be executed repeatedly in the future. Queries such as those used to generate periodic reports, perform routine summarization and data analysis belong to this category. Repetitive queries can constitute a large portion of the daily activities of a database system, and thus deserve extra optimization efforts. In this paper, we propose to record information about how tuples are joined in a repetitive query, called the *query trace*. We prove that the query trace is sufficient to compute the exact selectivities of joins for all plans of a given query. To reduce the space and time overheads in generating the query trace, we propose to construct only a sample of the query trace, called a *sample trace*, which can be much smaller than a (complete) query trace. A special operation, called a *sample outer join*, is designed to accomplish this feat. Accurate estimations of join selectivities, with associated confidence intervals, can be derived easily using the sample trace. Extensive experiments show that the sample trace can be constructed efficiently and be a controllable trade-off between accuracy and efficiency in estimations of join selectivities for repetitive queries.**

Keywords-query optimization, query re-optimization, trace, sampling method, sample trace

I. INTRODUCTION

Query re-optimization aims to refine execution plans of queries. There has been some progress made on this subject recently. In the literature, some [1–3], have focused on refining execution plans on-the-fly for currently running queries, while others [4–8], etc., on refining cost estimation for future queries. In this paper, we are interested in identifying useful statistics for refining cost estimation for future queries, similar to the latter.

To refine cost estimation for future queries, a common approach is to collect actual selectivities [7], [8], and some other statistics of the operators [5] and use them to adjust cost estimation for future queries. Unfortunately, selectivities of joins obtained from one plan of a query may not be sufficient for estimating selectivities of joins of another plan of the same query because as the join order changes, the selectivities of joins change, not to mention the selectivities of joins of other queries. The problem is exacerbated by adding selection predicates to the queries (to specify the tuples of interest). It is probably difficult to gather all information that may be needed for estimating selectivities of joins for all plans of all possible queries. Therefore, in this research, we set a more realistic goal by restricting ourselves to considering only a subset, but a large and frequently used subset, of queries, called repetitive queries.

Repetitive queries refer to those that are likely to be posted repeatedly in the future. Many useful queries, such as those used for generating periodical reports, performing routine maintenances, summarizing and grouping data for analysis, are repetitive queries. They are often stored in databases for convenient reuses for the long term. Any sub-optimality in the execution plans of such queries may mean repetitive and continued waste of system resources and time. Moreover, as new queries are continuously being developed, more queries become repetitive queries. Usually, the longer a database is in production, the greater the number of repetitive queries is in the database; their executions can constitute a large part of daily activities of a database system. The optimality of execution plans of repetitive queries has a tremendous effect on the performance of the system and thus deserves more optimization efforts.

Unlike much of the existing re-optimization work that focuses on refining physical execution plans of queries, our research focuses on refining logical execution plans (or the join order) of queries. Selectivities obtained from previous executions can be very useful for selecting an efficient access method (e.g., table scan and index access) and join method (e.g., nested-loop, sort-merge, etc.), that is, physical plans, but they may not be sufficient to estimate the selectivities of joins of the query in other join orders accurately without using simplifying assumptions, such as attribute independence and/or distribution uniformity. Our approach here focuses on how to gather sufficient information for computing the selectivities of joins for all plans of a query accurately.

In this paper, we continue to study the query or join trace [9], [10] that records information about how tuples are joined in the query. We have designed operators to gather such information so that the exact join selectivities in all join orders for a query can be computed. To reduce the overheads incurred in collecting a query trace, we design an innovative sample scheme that generates only a sample of the query trace. The sampling scheme is very effective in reducing overheads and the experimental results have shown that the sample trace is very accurate. With accurate selectivity estimations, running repetitive queries in the most efficient ways becomes possible.

The rest of the paper is organized as follows. Section II discusses previous work in relation to re-optimization. Section III introduces the query trace. The relationships between the traces and selectivities of queries with acyclic join graphs are

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-334-6

discussed in Sections IV. In Section V, we discuss deriving a sample of the trace and using the sample trace to estimate the selectivity of an arbitrary subquery. The sample trace is empirically evaluated in Section VI. Section VII presents the conclusions and future work.

II. LITERATURE SURVEY

The work in query re-optimization can be classified into two categories: (1) re-optimizations of current (or ongoing) queries and (2) re-optimization of future queries. Our work falls into the second category as we try to optimize the future executions of repetitive queries.

There has been much work that falls into the first category. ReOpt [3] discussed re-optimization of join queries. Statistics are collected at run-time and ad hoc heuristics are used to determine whether or not to re-optimize by either changing the execution plan or improving the resource allocations for the remainder of the execution. POP [11] improved upon ReOpt [3] by computing a more rigorous validity range for an input to a join within which the plan is valid. When the actual cardinality falls outside the validity range, a re-optimization is triggered. Rio [1] further improved on POP by computing interval estimates, instead of point estimates, of the cardinalities. Within the intervals, they selected robust and switchable plans to avoid repeated re-optimization and loss of earlier pipelined work.

The second category includes [4], [11–15] Chen et al. [14] first used query feedback to refine the data distribution, represented by a linear combination of "model function". Only 1-dimensional distributions were considered. Aboulnaga et al. [4] used the actual range selectivities obtained from queries to adjust histograms. By splitting high frequency buckets and merging similar consecutive buckets, histograms are tuned. Lim et al. [6] used actual selectivities to tune bucket frequencies and query workloads to determine what set of histograms to build. All these approaches are mainly concerned with selection queries.

The pay-as-you-go approach [16] improves the idea of LEO [7], [8] with proactive monitoring and plan modifications. Nevertheless, it does not collect enough information needed for estimating arbitrary logical execution plans, and it must rely on repetitive executions of the query to collect complementary cardinality information.

Microsoft Index Wizard [17] recommends the database administator (DBA) on indexes. It is similar to DB2 Advisor [18] and Oracle SQL Access Advisor [19] that are limited to access path recommendations rather than selections of logical execution plans.

Xplus [20] enumerates plans and their neighborhoods to search for alternative plans with lower cost. However, it may only find plans with local minimum values that may not necessarily be the optimal plan.

Oracle's Automatic Tuning Optimizer (ATO) [21] performs SQL Profiling and what-if analysis on selected high load SQL statements. SQL Profiles are used with other statistics to build a well-tuned plan. However, it may not have sufficient statistics that are needed for cost estimation of all plans of the query, and must generate auxiliary information for estimations, which can be time-consuming and inaccurate.

In this research, we are interested in gathering sufficient information about a query during execution so that an optimizer can use the information to find the best join order, or the best logical execution plan, for the query. Existing re-optimiation works, such as POP, Rio, and LEO can be very effective in refining physical execution plans of queries, e.g., access methods (e.g., table scan, and index access and join methods (e.g., nested loop, sort-merge, and hash join using the actual selectivities obtained from the feedback. However, they may fall short of optimizing logical execution plans of the queries because as the join orders change, the selectivities of joins in alternative plans change too. It requires more information than just the selectivities of operators of the current plan to estimate the selectivities of joins of alternative plans accurately.

III. QUERY TRACE

When a query is being processed, information about how tuples are joined is gathered. We intend to use this information, called the query or join trace, to estimate selectivities of joins in all execution orders.

We use tuple IDs to identify matching tuples in the joins of a query in the trace. To create the trace of a query, an ID attribute, R_i-ID, is added to (the schema of) each relation R_i. The added ID attributes are to be included in the output (schema) of all operations to identify tuples that contribute to the output. For example, in the join of two relations R_1 and R_2, a result tuple, besides its normal set of attributes, will have two additional attributes: R_1-ID and R_2-ID, whose values identify the pairs of tuples that match in the join of R_1 and R_2. It is noted that we do not need to add an ID attribute physically to (the schema of) a relation on disk, but just append an ID value when a tuple flows into the join operation.

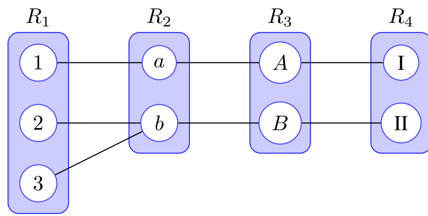
In the following, we use an example to show how query traces look like in their simplest form and how they are generated. More complicated examples will be discussed in subsequent sections, including using other operators, such as outer joins, or designing new operators, such as extended outer joins, to gather sufficient information in the trace for different types of queries.

Example 1. (**Trace**) Consider a left-deep tree execution plan $P = ((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$. To generate the trace, an ID attribute is added to every relation and the attribute is to be preserved in the outputs of all operators. Thus, the result of $R_1 \bowtie R_2$, as shown in Fig. 1(b), besides its normal set of attributes, denoted by Result-Attrs, has additional attributes R_1-ID and R_2-ID, called the *trace* of $R_1 \bowtie R_2$, denoted by $T(R_1 \bowtie R_2)$.

Once a query is completely processed, we can extract the final trace, e.g., $T(((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4)$ in Example 1. Keeping the final trace is enough to derive the selectivities of joins in all execution orders Consequently, we shall retain only the final trace for selectivity re-estimation. *Hereafter, the trace of a query refers to the final trace of the query, unless otherwise stated.*

IV. SELECTIVITY ESTIMATION FOR ACYCLIC JOIN GRAPHS USING TRACE

Let Q be a query with an acyclic join graph $G(V, E)$ and P an execution plan of the query. Let $T(P)$ be the final trace of P. Let $G'(V', E')$ be a vertex-induced connected subgraph of $G(V, E)$, where $V' = \{R_{i_1}, ..., R_{i_m}\} \subseteq V$ and $E' \subseteq E$, representing a subquery Q' of Q. We propose to compute the exact selectivity of Q' as



(a) Matching of Tuples

Result-Attrs	R_1-ID	R_2-ID
...	1	a
...	2	b
...	3	b

(b) Result and Trace of $R_1 \bowtie R_2$

R_1-ID	R_2-ID	R_3-ID
1	a	A
2	b	B
3	b	B

(c) Trace of $(R_1 \bowtie R_2) \bowtie R_3$

R_1-ID	R_2-ID	R_3-ID	R_4-ID
1	a	A	I
2	b	B	II
3	b	B	II

(d) Trace of $((R_1 \bowtie R_2) \bowtie R_3) \bowtie R4$

Figure 1. Query Traces with Tuple IDs

$$\widetilde{sel}(Q') = \frac{|\pi_{R_{i_1}\text{-ID},...,R_{i_m}\text{-ID}} T(P)|}{|R_{i_1}| \times ... \times |R_{i_m}|} \quad (1)$$

where $\pi_{R_{i_1}\text{-ID},...,R_{i_m}\text{-ID}} T(P)$ is the projection of trace $T(P)$ on attributes R_{i_1}-ID, ..., R_{i_m}-ID, with duplicates removed.

As we shall discuss later that trace tuples, generated by other operators (e.g., (extended) outer joins), could have null values in some of the projected ID components R_{i_1}-ID, ... ,R_{i_m}-ID, and such tuples shall not be accounted for in $|\pi_{R_{i_1}\text{-ID},...,R_{i_m}\text{-ID}} T(P)|$

The cardinalities of the input relations $|R_{i_1}|, ..., |R_{i_m}|$ can be obtained easily. If $R_{i_j}, 1 \leq j \leq m$, is a base relation, $|R_{i_j}|$ is certainly known. If R_{i_j} represents a relation that is immediately preceded by some selections and projection, $|R_{i_j}|$ can be obtained by counting the number of tuples flowing into the join operation. Another possible way to compute $|R_{i_j}|$ is just to count the numbers of unique IDs in the respective ID columns in the final trace. Thus, without regard to the access methods used to retrieve tuples from relations, such as table scan, index access, etc., exact $|R_{i_j}|$ can always be obtained.

A. No Dangling Tuples in the Joins

Here, we assume no dangling tuple exists in any of the joins in the query. The join relationships in Fig. 1(a) satisfy this condition. Now, let us see how accurate (1) derives the selectivities.

Example 2. (No Dangling Tuple). Consider the query and plan in Example 1. Given the (final) query trace in Fig. 1(c), the selectivities of $R_1 \bowtie R_2$, $R_2 \bowtie R_3$, and $R_3 \bowtie R_4$, are derived, by (1), as $\frac{1}{2}$, $\frac{1}{2}$, and $\frac{1}{2}$, respectively, which are the exact selectivities of the respective joins. The derived selectivities of $(R_1 \bowtie R_2) \bowtie R_3 (= (R_2 \bowtie R_3) \bowtie R_1 = (R_3 \bowtie R_2) \bowtie R_1), (R_3 \bowtie R_4) \bowtie R_2 (= (R_3 \bowtie R_2) \bowtie R_4 = (R_4 \bowtie R_3) \bowtie R_2)$, are all 1/4; again they are all exact.

It is not a coincidence that the estimated selectivities are exact. In fact, we can prove that if there is no dangling tuple in any join of the query, (1) derives the exact selectivities of all possible subqueries.

Theorem 1 (Exact Estimation without Dangling Tuples). *Let P be an execution plan of a query Q with a connected acyclic join graph $G(V, E)$. Let Q' be a subquery of Q that has a vertex-induced connected join subgraph $G'(V', E')$, $V' = \{R_{i_1}, ..., R_{i_m}\} \subseteq V$. If there is no dangling tuple in any join of P, (1) gives the exact selectivity of Q' from $T(P)$.*

Proof: Please see Appendix A. ■

B. Dangling Tuples in Joins

Now, let us consider joins with dangling tuples. Dangling tuples are lost in the joins. To retain matching information about dangling tuples, we replace the joins in the original query by the full outer joins ($\overset{\circ}{\bowtie}$). Fig.2(c) to 2(e) show the traces generated at different stages of query execution, where the joins are replaced by the full outer joins. The trace in Fig. 2(c) is the same as if it were generated by a join because there is no dangling tuple in the join. The trace in Fig. 2(d) retains information about dangling tuples b in R_2 and B in R_3 by the outer join.

The estimated selectivities for $R_1 \bowtie R_2$, $R_2 \bowtie R_3$, and $R_3 \bowtie R_4$ are now, by (1), $\frac{1}{2}(= \frac{2}{2\times2})$, $\frac{1}{4}(= \frac{1}{2\times2})$, and $\frac{1}{2}(= \frac{2}{2\times2})$, respectively, which are exact. Note that, as mentioned earlier, a trace tuple having a null for any of the projected attributes is not accounted for in the respective $|\pi_{R_{i_1}\text{-ID},...,R_{i_1}\text{-ID}} T(P)|$ because a null in a R_{i_j}-ID column of a trace tuple indicates that there is no match found in R_{i_j} for the respective combination of tuples to generate an output in the (sub)query. One can easily verify that the estimated selectivities for all other subqueries are all exact.

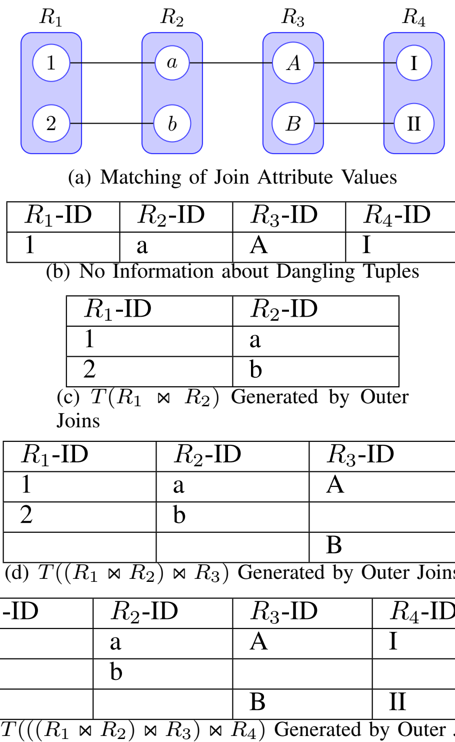
Theorem 2 (Exact Estimation with Dangling Tuples). *Let P be an execution plan of a query Q with a connected acyclic join graph $G(V, E)$. Let Q' be a subquery of Q that has a vertex-induced connected join subgraph $G'(V', E')$, $V' = \{R_{i_1}, \ldots, R_{i_m}\} \subseteq V$. (1) gives the exact selectivity of Q' from the trace obtained by replacing the joins in the query with the full outer joins, denoted by $T(P)$.*

Proof: Please see Appendix B. ■

For simplicity, hereafter an outer join refers to a full outer join, unless otherwise stated. Note also that we have used $T(P)$ to denote the trace of a query, regardless of whether the trace is generated by the joins, outer joins, or even other operators (to be discussed shortly).

V. Sample Trace and Selectivity Estimation

Dangling tuples are retained in the outer and extended outer joins. Thus, the result trace can contain more tuples than the query result. This situation is exacerbated when relations are preceded by selection predicates, which can eliminate matching tuples from operand relations. In this section, we discuss a sampling design that not only can significantly reduce the size of result trace, but it also can provide accurate selectivity estimations.



(a) Matching of Join Attribute Values

R_1-ID	R_2-ID	R_3-ID	R_4-ID
1	a	A	I

(b) No Information about Dangling Tuples

R_1-ID	R_2-ID
1	a
2	b

(c) $T(R_1 \bowtie R_2)$ Generated by Outer Joins

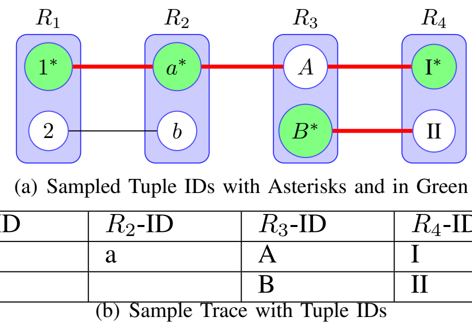
R_1-ID	R_2-ID	R_3-ID
1	a	A
2	b	
		B

(d) $T((R_1 \bowtie R_2) \bowtie R_3)$ Generated by Outer Joins

R_1-ID	R_2-ID	R_3-ID	R_4-ID
1	a	A	I
2	b		
		B	II

(e) $T(((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4)$ Generated by Outer Joins

Figure 2. Dangling Tuples in Relations



(a) Sampled Tuple IDs with Asterisks and in Green

R_1-ID	R_2-ID	R_3-ID	R_4-ID
1	a	A	I
		B	II

(b) Sample Trace with Tuple IDs

Figure 3. An Example of Sample Trace

1: **for** each tuple $t_r \in R_r$ **do**
2:   **for** each match $t_l \in R_l$ **do**
3:     output a result tuple t
4:     tag[t]= tag[t_l] OR tag[t_r]
5:     **if** no match found in R_l and tag[t_r]=1 **then**
6:       output a result tuples with nulls for all attributes of R_r
7:     **end if**
8:   **end for**
9: **end for**
10: **for** each tuple $t_l \in R_l$ that found no match in R_r and tag[t_l]=1 **do**
11:   Output a result tuple with null for all attributes of R_l.
12: **end for**

Figure 4. Algorithm of Sample Outer-join

A. Sample Trace

A set of uniform random sample tuples is designated for each relation. Instead of retaining all dangling tuples in the trace, we keep only those dangling tuples that are related to sample tuples. Certainly, query result tuples must be kept as before. This partial trace is called a **Sample Trace**.

Example 3. (Sample Trace) Consider Fig. 3(a). Assume tuple 1, a, B, and I (in green and with asterisk) are picked as sample tuples respectively from R_1, R_2, R_3, and R_4. We intend to generate trace tuples only for query result tuples and dangling tuples that are related to the sample tuples. In other words, only those trace tuples containing any of 1, a, B, I, or result tuples will be generated. That is, only the first and third rows of the original trace (red lines in Fig. 3(a)) will be generated to be the *Sample Trace*, as shown in Fig. 3(b).

Definition 1. (**Sample Trace**) Given a join query Q with the plan $(((R_1 \bowtie R_2)... \bowtie R_i)... \bowtie R_n)$, and $\{S_i\}_{i=1}^n$ are the simple random samples of $\{R_i\}_{i=1}^n$. The **Sample Trace** of Q, T^*, is the subset of the trace tuples of Q that are generated by sample tuples from $\{S_i\}_{i=1}^n$.

To generate such a sample trace, the outer join operator must be modified. The modified operator is called a **Sample Full Outer Join** or just a **Sample Outer Join**. If the input tuple is a sampled tuple or its joinable with a sample tuple, then the sample outer join performs same as an outer join; otherwise, it performs like the ordinary join operation. That is, a dangling tuple in a join will become a result tuple of the sample outer join only if it is a sample tuple from a base relation or it is an intermediate result tuple derived from a sample tuple. Certainly, a pair of match tuples generates an output tuple in the sample outer join, like in an ordinary join.

Figure 4 describes the procedure of Sample Outer-join (R_l, R_r). Let R_l and R_r be the left and right operand relations of a sample outer join. A boolean tag tag[t] is associated with each tuple t, with tag[t]=1 indicating that t is a sample tuple or is derived from a sample tuple; tag[t]=0, otherwise.

B. Selectivity Estimation

Consider a query Q with the plan $(((R_1 \bowtie R_2)... \bowtie R_i)... \bowtie R_n)$. Let T^* be its sample trace generated by replacing the joins with the sample outer joins. Given a subquery q involving R_{i_1},..., R_{i_j}, ... , R_{i_k}, $1 \leq j \leq k$, we attempt to estimate the result size of q using T^*. More precisely, we will use only a subset of T^* that is related to the sample tuples from R_{i_1}, to estimate the query size.

Let S_{i_1} be the set of IDs of sample tuples from R_{i_1}. We denote the subset of T^* that is related to the sample tuples from R_{i_1} by $T^*_{i_1}$, that is, $T^*_{i_1} = \sigma_{R_{i_1}\text{-ID} \in S_{i_1}}(T^*)$. Let n_{i_1} be the number of result tuples of q generated by the sample tuples from R_{i_1}. It can be observed that

$$n_{i_1} = \left| \prod_{R_{i_1}-ID,...,R_{i_k}-ID} T^*_{i_1} \right| \tag{2}$$

where $\prod$ is a projection operation. As mentioned, a tuple with a null in any of the attributes R_{i_1}-ID,..., R_{i_k}-ID will be removed in the above projection. Our estimation formula for the subquery q is stated in the following theorem.

Theorem 3 (**Unbiased Estimation with Sample Trace**). *Consider a query Q with the plan* $(((R_1 \bowtie R_2)... \bowtie R_i)... \bowtie R_n)$, *and a subquery q involving* R_{i_1},..., R_{i_j}, ... , R_{i_k}, $1 \leq j \leq k$.

Copyright (c) IARIA, 2014.

The query result size of q, Y_q, is estimated as

$$\widehat{Y}_q = n_{i_1} \frac{|R_{i_1}|}{|S_{i_1}|} \tag{3}$$

which is an unbiased estimator of the query result size of subquery q. Here, $|R_{i_1}|$ and $|S_{i_1}|$ are the sizes of R_{i_1} and S_{i_1}, respectively. And n_{i_1} is described in (2).

Proof: Please see Appendix C. ■

By Theorem 2.2 in [22], we get the variance of the query estimation using sample trace in the following theorem.

Theorem 4 (Variance of Estimator). *Let*

$$S^2 = \frac{\sum_{j=1}^{|R_{i_1}|}(Y_j - \overline{Y}_q)^2}{|R_{i_1}| - 1}$$

in which Y_j is the total number of result tuples generated by the jth tuple in relation R_{i_1} in subquery q, and $\overline{Y}_q = \frac{1}{|R_{i_1}|}\sum_{j=1}^{|R_{i_1}|} Y_j$. The variance of the query size estimation for subquery q using sample trace is

$$Var(\hat{Y}_q) = \frac{|R_{i_1}|^2}{|S_{i_1}|} S^2 \left(1 - \frac{|S_{i_1}|}{|R_{i_1}|}\right)$$

We can also get an unbiased estimation of $Var(\hat{Y}_q)$ from the values in the sample relations. By Theorem 2.4 in [22], we have the following theorem.

Theorem 5 (Approximation of Variance). *Let*

$$s^2 = \frac{\sum_{j=1}^{|S_{i_1}|}(y_j - \overline{y})^2}{|S_{i_1}| - 1}$$

in which y_j is the total result tuples generated by the jth tuple in the sample relation S_{i_1}, and $\overline{y} = \frac{n_{i_1}}{|S_{i_1}|}$ An unbiased estimation of $Var(\hat{Y}_j)$ is

$$v(\hat{Y}_q) = \frac{|R_{i_1}|^2}{|S_{i_1}|} s^2 \left(1 - \frac{|S_{i_1}|}{|R_{i_1}|}\right)$$

It is usually assumed that the estimated value $\widehat{Y}_q$ is normally distributed about the corresponding true query result size Y_q. When this assumption holds, by Central Limit Theory [22], we derive the confidence interval of Y_q as follows.

Theorem 6 (Confidence Interval of Y_q). *The associated confidence interval of Y_q can be computed as*

$$\left(\hat{Y}_q - t_p\sqrt{v(\hat{Y}_q)}, \hat{Y}_q + t_p\sqrt{v(\hat{Y}_q)}\right)$$

where $v(\hat{Y}_q)$ is in Theorem 5 and t_p is the normal deviate corresponding to the desired probability p.

VI. EXPERIMENTAL RESULTS

The experiments will focus on the feasibility of using sample traces by examining their estimation accuracy and construction overheads, including space and time. We replace the joins in a query by sample outerjoins to construct the sample trace and use it to estimate the selectivities of joins in all possible orders for the query. All programs are implemented in Java on a X86 Linux Desktop, equipped with a 3.4 GHz CPU, a 4GB RAM, and a 500GB hard drive (7200rpm, buffer size 16MB). All data, including datasets, test queries, and intermediate query results, is materialized on a local hard drive. To facilitate the evaluation of sample outerjoins, we use index files.

A. Datasets and Sampling Ratios

We conducted experiments on both synthetic data — the TPC-H skewed datasets [23] , and real-life data — the DBLP dataset [24]. We generated 1GB TPC-H datasets with different skew factors of z =0.5, 1, and 1.5. Each TPC-H dataset has 8 relations. The DBLP database listed more than 1.3 million articles in computer science. The DBLP dataset sources from a relational enhancement of the original DBLP data, named DBLP++ [25]. The many-many relationship between "author" and "paper" has been replaced by introducing an "author_of" relation and two many-one relationships: from "author_of" to "author", and "author_of" to "paper", as it is often did in relational databases.

Sampling ratio indicates the fraction of tuples that are sampled to construct the sample trace. We performed experiments with sampling ratios from 0.1%, 0.2%, ..., to 1% to test the average performance, as shown in Fig. 5.

B. Test Queries

A set of test queries is constructed for each dataset. For the TPC-H datasets, we chose a subset of the original TPC-H benchmark queries that contain joins of 3 to 8 selections. The total number of queries is 15 from which we construct 50 subqueries, with different join orders, to examine the accuracy and overheads. For the DBLP dataset, we generated 5 queries that involved joins of all relations in the database. 20 subqueries are tested. Note that, selection predicates are deployed on relations. The ranges of selection predicates are randomly generated to examine the average performance of the sample traces under complex selection conditions.

C. Space Performance

Besides the query result tuples, the sample outer join retains dangling tuples that are related to the sample tuples in evaluation, which contributes to the space overhead.

The space overhead, denoted SO, of the sample trace is calculated as

$$SO = \frac{\#\{\text{intermediate dangling tuples}\}}{\#\{\text{intermediate query result sizes}\}} \times 100\%$$

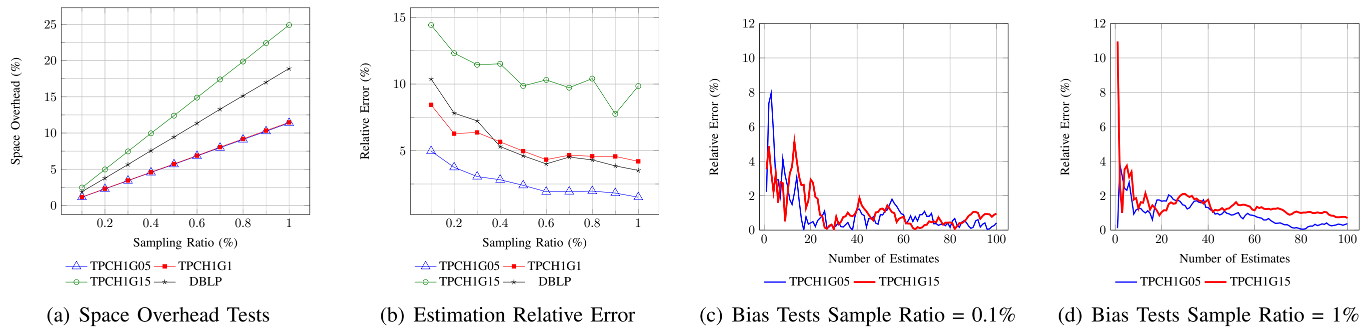
in which "#{}" denotes the cardinality of a set, and "intermediate" means that we account for all sample traces generated during the intermediate phases of a query.

The space overheads for datasets TPCH1G05 and TPCH1G1 are very close. It is because their skew factors are relatively low ($z = 0.5$ and 1). When the skew factor z increased to 1.5 (i.e., very skewed) in TPCH1G15, the space overhead in Fig. 5(a) grew noticeably higher than the other two low skewed TPC-H datasets ($z = 0.5$ and 1). Under the same selection ranges in the test queries, greater skew factor produced more dangling tuples, as explained earlier.

Results on the DBLP dataset are similar. The space overhead increased with the increase of the sampling ratio. The space overhead was affected by the skewness of the dataset and the selection criteria of the queries.

D. Accuracy Performance

The results of the accuracy tests are shown in Fig. 5(b). We use the absolute relative error, or relative error, E, to evaluate



(a) Space Overhead Tests (b) Estimation Relative Error (c) Bias Tests Sample Ratio = 0.1% (d) Bias Tests Sample Ratio = 1%

Figure 5. Sample Trace Tests on Multiple Datasets

TABLE I. AVERAGE PERFORMANCE

	TPCH1G05	TPCH1G1	TPCH1G15	DBLP
Space Overhead (%)	6.26	6.32	13.68	10.40
Construction Time (s)	0.06	0.06	0.05	0.30
Time Overhead (%)	1.10	1.12	1.06	5.90
Relative Error (%)	2.62	5.41	10.76	5.56

the accuracy of using a sample traces, which is defined as follows.

$$E = \left| \frac{\widehat{Y}^{Est} - Y^{Act}}{Y^{Act}} \right| \times 100\%$$

where Y^{Act} denotes the actual query result size and $\widehat{Y}^{Est}$ is the estimated result size.

To evaluate the average performance, for each test query, we generated 10 sets of sample traces, and use them to estimate the sizes of the subqueries of these test queries.

As shown in Fig. 5(b), the relative errors of all datasets decreased gradually as the sampling ratio increased. The relative errors for datasets TPCH1G05 ($z = 0.5$), TPCH1G1 ($z = 1.0$), and DBLP decreased more smoothly than the relative errors for TPCH1G15 ($z = 1.5$), which has a higher skew factor than others. In general, it is more difficult to represent a skewed distribution than a smooth distribution. Therefore, the more skewed the data, the higher the sampling ratio is needed to achieve the same accuracy.

E. Average Performance

The average performances of sample trace on all datasets are listed in Table I. Note that the minimum sampling ratio is 0.1% and the maximum is 1%. The space overheads are less than 13.68%, and construction times are no more than 0.3 seconds. In addition, under a low sampling ratio, the maximum relative error is 10.76%, which occurred on TPCH1G15 dataset when the sampling ratio is 0.1%.

In summary, with a small amount of samples (e.g., $\leq$ 1%), accurate estimation of subquery result sizes (e.g., $\leq$ 10%) can be obtained, even for highly skewed data. And the time and space overheads are mostly small, around 1% and 10%, respectively. The experimental results confirm that sample trace can be a viable approach for re-estimation of selectivities for repetitive queries. It is effective and accurate.

F. Bias Test of the Sample Trace Estimator

In Section V, we proposed using $\widehat{Y}_q$ to estimate a subquery with sample trace, which is proved to be unbiased in Theorem 3. Here we empirically validate the unbiasedness of $\widehat{Y}_q$. For demonstrative purpose, we choose one query and one of its subqueries from previous test queries. We performed the unbiased tests on two datasets TPCH1G05 and TPCH1G15, where the sample ratios are equal to 0.1% and 1%, respectively. A subquery is estimated using a different sample trace of the same sample ratio. We generated 100 sample traces for this purpose. To demonstrate the variation tendency of bias when the number of estimations increases, we average the first i estimation values by absolute average relative error defined as

$$E_i = \left| \frac{\frac{\sum_{j=1}^{i} \widehat{Y}_{q_j}^{Est}}{i} - Y_q^{Act}}{Y_q^{Act}} \right| \times 100\%$$

where $i \geq 1$, Y_q^{Act} is the actual subquery size, $\widehat{Y}_{q_j}^{Est}$ is the estimated value derived from the jth sample trace. For an unbiased estimator, the absolute average relative error should approach 0 as i (the number of tries) increases [22].

As observed in Fig 5(c) and Fig 5(d), the errors on both TPCH1G05 and TPCH1G15 approached 0 as the number of estimations increased. The experiments validate the earlier proof of the unbiasedness of the sample trace estimator. Note that when the skew factor is relatively higher, in TPCH1G15, the relative errors are generally higher. Also, the error curves in Fig 5(d) converged smoother than those in Fig 5(c), where the sample ratios are equal to 1% and 0.1%, respectively.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed utilizing information about how tuples are matching in the joins in a query, called the query trace, and estimate selectivities of all subqueries of a repetitive query. To gather sufficient information in the traces, we used the full outerjoins for queries with acyclic join graphs. We have shown that the exact selectivities of joins in all execution orders of the query can be computed from its trace.

To reduce the overheads incurred in collecting traces, we enhanced the outerjoins with sampling capability so that a random sample of the trace can be obtained efficiently. Experimental results have shown that with a small amount of sample tuples, accurate selectivity estimations can be obtained. Sample traces can be a very efficient and effective tool for the re-optimization of repetitive queries.

For future work, we would like to give an estimated sample size to get a satisfying estimation confidence interval with

efficient space budget. We are also interested in, for highly skewed data, how to use adaptive sampling on the relations to construct the adaptive sample trace.

References

[1] S. Babu, P. Bizarro, and D. DeWitt, “Proactive re-optimization,” SIGMOD ’05, (New York, NY, USA), ACM, 2005, pp. 107–118.

[2] N. Kabra and D. J. DeWitt, “Efficient mid-query re-optimization of sub-optimal query execution plans,” SIGMOD ’98, (New York, NY, USA), ACM, 1998, pp. 106–117.

[3] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic, “Robust query processing through progressive optimization,” SIGMOD ’04, 2004, pp. 659–670.

[4] A. Aboulnaga and S. Chaudhuri, “Self-tuning histograms: building histograms without looking at data,” SIGMOD Rec., vol. 28, June 1999, pp. 181–192.

[5] S. Chaudhuri, V. Narasayya, and R. Ramamurthy, “Diagnosing estimation errors in page counts using execution feedback,” (Washington, DC, USA), IEEE Computer Society, 2008, pp. 1013–1022.

[6] L. Lim, M. Wang, and J. S. Vitter, “Sash: a self-adaptive histogram set for dynamically changing workloads,” VLDB ’2003, VLDB Endowment, 2003, pp. 369–380.

[7] V. Markl, G. M. Lohman, and V. Raman, “LEO: An autonomic query optimizer for DB2,” IBM Syst. J., vol. 42, Jan. 2003, pp. 98–106.

[8] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil, “LEO - DB2’s learning optimizer,” VLDB ’01, (San Francisco, CA, USA), 2001, pp. 19–28.

[9] F. Yu, W.-C. Hou, C. Luo, Q. Zhu, and D. Che, “Join selectivity re-estimation for repetitive queries in databases,” in Database and Expert Systems Applications, vol. 6861 of Lecture Notes in Computer Science, pp. 420–427, 2011.

[10] F. Yu, Constructing Accurate Synopses for Database Query Optimization and Re-optimization. PhD thesis, Carbondale, IL, USA, 2013.

[11] K. Ono and G. M. Lohman, “Measuring the complexity of join enumeration in query optimization,” VLDB ’97, (San Francisco, CA, USA), 1990, pp. 314–325.

[12] N. Bruno, S. Chaudhuri, and L. Gravano, “Stholes: a multidimensional workload-aware histogram,” SIGMOD ’01, (New York, NY, USA), ACM, 2001, pp. 211–222.

[13] S. Christodoulakis, “Implications of certain assumptions in database performance evauation,” ACM Trans. Database Syst., vol. 9, June 1984, pp. 163–186.

[14] C. M. Chen and N. Roussopoulos, “Adaptive selectivity estimation using query feedback,” SIGMOD ’94, (New York, NY, USA), ACM, 1994, pp. 161–172.

[15] V. Poosala and Y. E. Ioannidis, “Selectivity estimation without the attribute value independence assumption,” VLDB ’97, 1997, pp. 486–495.

[16] S. Chaudhuri, V. Narasayya, and R. Ramamurthy, “A pay-as-you-go framework for query execution feedback,” Proc. VLDB Endow., vol. 1, Aug. 2008, pp. 1141–1152.

[17] S. Chaudhuri and V. R. Narasayya, “An efficient cost-driven index selection tool for microsoft sql server,” VLDB ’97, (San Francisco, CA, USA), 1997, pp. 146–155.

[18] G. Valentin, M. Zuliani, D. C. Zilio, G. Lohman, and A. Skelley, “DB2 advisor: An optimizer smart enough to recommend its own indexes,” ICDE ’00, (Washington, DC, USA), IEEE Computer Society, 2000, pp. 101–.

[19] O. W. Paper, “Oracle coporation: Performance tuning using the sql access advisor.” http://otn.oracle.com, 2003.

[20] H. Herodotou and S. Babu, “Xplus: a sql-tuning-aware query optimizer,” Proc. VLDB Endow., vol. 3, Sept. 2010, pp. 1149–1160.

[21] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin, “Automatic sql tuning in oracle 10g,” VLDB ’04, VLDB Endowment, 2004, pp. 1098–1109.

[22] W. G. Cochran, Sampling Techniques, 3rd Edition. John Wiley, 1977.

[23] S. Chaudhuri and V. Narasayya, “Program for tpc-d data generation with skew..” ftp://ftp.research.microsoft.com/users/viveknar/tpcdskew.

[24] M. Ley, “The DBLP computer science bibliography.” http://www.informatik.uni-trier.de/~ley/db/.

[25] J. Diederich, “FacetedDBLP.” http://dblp.l3s.de/dblp++.php. [Retrieved Dec, 2013].

Appendix

A. Proof of Theorem 1

Proof: We show that there is a one-to-one correspondence between a result tuple of $\pi_{R_{i_1}\text{-ID},\ldots,R_{i_m}\text{-ID}}T(P)$ and a result tuple of Q'.

“$\Rightarrow$” By the construction of the trace, for each edge (R_i, R_j) in E, the R_i-ID and R_j-ID record the matches in $T(P)$. Let S be a set of tuples $\{t_{i_1}, \ldots, t_{i_m}\}, t_{i_j} \in R_{i_j} \in V'$, whose IDs appear in a result tuple of $\pi_{R_{i_1}\text{-ID},\ldots,R_{i_m}\text{-ID}}T(P)$. For each edge $(R_{i_j}, R_{i_k}) \in E', 1 \leq j, k \leq m$, t_{i_j} and t_{i_k} must match in their join attribute values because their IDs appear in the same trace tuple. That is, the set of tuples S satisfies all the join predicates placed between relations in V' and thus can generate a result tuple in Q' by joins. Moreover, the joins of $\{t_{i_1}, \ldots, t_{i_m}\}$ cannot generate more than one tuple.

“$\Leftarrow$” Let $S' = \{t'_{i_1}, \ldots, t'_{i_m}\}, t'_{i_j} \in R_{i_j} \in V'$ be a set of tuples that generates a result tuple in Q'. With joinable tuples from $V - V'$ (they always exist because there is no dangling tuples in any join), S', following plan P, can generate at least one result tuple in Q. Since S' has no null tuple, their corresponding IDs must appear as a result tuple of $\pi_{R_{i_1}\text{-ID},\ldots,R_{i_m}\text{-ID}}T(P)$. The projection $\pi_{R_{i_1}\text{-ID},\ldots,R_{i_m}\text{-ID}}$ will retain only one set of IDs corresponding to $\{t'_{i_1}, ..., t'_{i_m}\}$ in $\pi_{R_{i_1}\text{-ID},\ldots,R_{i_m}\text{-ID}}T(P)$. ■

B. Proof of Theorem 2

Proof: We show that there is a one-to-one correspondence between a result tuple of $\pi_{R_{i_1}\text{-ID},\ldots,R_{i_m}\text{-ID}}T(P)$ and a result tuple of Q'.

“$\Rightarrow$” Same arguments as in $\Rightarrow$ of Theorem 1.

“$\Leftarrow$” Let $S' = \{t'_{i_1}, ..., t'_{i_m}\}, t'_{i_j} \in R_{i_j} \in V'$ be a set of tuples that generates a result tuple in Q’. With joinable tuples or null tuples (if there are no joinable tuples) from the relations $V - V'$, S', following plan P, can generate at least one result tuple in the query derived from Q in which all joins are replaced by full outer joins; and all these result tuples must have valid IDs for all R_i-ID attributes, $R_i \in V'$, since there is no null tuple in S'. Since duplicates are eliminated in $\pi_{R_{i_1}\text{-ID},\ldots,R_{i_m}\text{-ID}}(T(P))$, S' can only produce one result tuple in $\pi_{R_{i_1}\text{-ID},\ldots,R_{i_m}\text{-ID}}(T(P))$. ■

C. Proof of Theorem 3

Proof: First of all, we denote the query result size of q by Y_q. The total result tuples in the subquery q generated by the jth tuple of R_{i_1} is denoted by n_j. i.e. In relation R_{i_1}, its jth tuple generated n_j result tuples in the final result of subquery q. The query result size of q equals the sum of result tuples generated by each tuple on relation R_{i_1}. Thus $Y_q = \sum_{j=1}^{|R_{i_1}|} n_j$. Also note that S_{i_1} is a simple random sample without replacement of R_{i_1}. And n_{i_1} is the sum of result tuples generated by S_{i_1}. By Theorem 2.1 of [22], we have $\widehat{Y}_q$ is an unbiased estimator of Y_q. ■