Rule Profiling for Query Optimizers and their Implications

Surajit Chaudhuri[1], Leo Giakoumakis[2], Vivek Narasayya[3], Ravishankar Ramamurthy[4]

Microsoft Corp
One Microsoft Way Redmond WA
{[1]surajitc, [2]leogia, [3]viveknar, [4]ravirama}@microsoft.com

Abstract—**Many modern optimizers use a transformation rule based framework. While there has been a lot of work on identifying new transformation rules, there has been little work focused on empirically evaluating the effectiveness of these transformation rules. In this paper we present the results of an empirical study of "profiling" transformation rules in Microsoft SQL Server using a diverse set of real world and benchmark query workloads. We also discuss the implications of these results for designing and testing query optimizers.**

I. INTRODUCTION

Query optimizers in today's DBMSs are responsible for obtaining a good execution plan for a given query. There has been extensive work on how to architect query optimizers in order to make them extensible. Many modern optimizers use a transformation rule-based framework. Examples include industrial query optimizers such as IBM Starbust [18], Microsoft SQL Server [10], Oracle [2], Tandem's NonStopSQL[1] as well as academic prototypes such as the Volcano optimizer [11]. Such optimizers use transformation rules as the basic primitive to generate different alternative plans for a query. There has also been a lot of work on identifying the set of transformation rules to be used by the optimizer and the set of transformation rules have grown significantly over time (see [4] for an overview). Some of the important classes of rules studied thus far include rules for pulling-up/pushing down a group-by operator (e.g.,[6][23]), handling nested sub-queries (e.g., [7][15][17][21]), commuting outer-joins with joins(e.g., [8]).

In spite of the fact that rule-based optimization is a well researched topic, there has been little work on empirically evaluating the effectiveness of these transformation rules. While the importance of profiling is well understood in general and such tools are an integral part of software development, surprisingly, there is a lack of work that is focused on "profiling" the query optimizer for its usage of transformation rules. Such profiling is of importance because modern optimizers rely on a large number (typically hundreds) of transformation rules and such rules directly impact the quality and performance of a query optimizer. As the number of optimizer rules has increased, there is serious concern that the plan choices of the optimizer could change unpredictably (e.g., even with small changes to selectivities). In fact, this concern has recently been reinforced by the study of Plan diagrams [20]. These diagrams were introduced as a way of visually explaining the complexity of the execution plan choices of a query optimizer and have demonstrated that the plan choices of current optimizers are perhaps too fine grained.

In this paper, we study the effectiveness of transformation rules in a commercial query optimizer (Microsoft SQL Server) for a variety of benchmark and real world queries. To the best of our knowledge, this is the first paper to do such profiling of transformation rules in a commercial strength optimizer.

Our first challenge in doing this study is to define what profiling means in the context of transformation rules. First, we examine the case of a single query and ask the question if we can identify the rules that are needed to obtain the optimal plan for the query. We refer to this as an *essential set* of rules for a particular query. It turns out that in some cases, there may be more than one viable essential set of rules for a query. That motivates the notion of a *relevant* transformation rule. A transformation rule is relevant for a query if it is part of every essential set of rules for that query. In other words, the optimal plan of a query will be missed unless all relevant transformation rules for a query are considered. These definitions naturally extend from a query to a *workload*, i.e. set of queries. These two key concepts provide the basis of our profiling.

The paper empirically studies these metrics for a variety of workloads. First, we observe that the essential set sizes for most queries are typically small (around 10% of the entire set of rules). Second, only a small number of rules are relevant across many queries in a workload. While simple to state, these results have potentially interesting implications for optimizer design and testing, as discussed below.

The search space of a query optimizer is largely determined by the set of transformation rules used by it. If we constrain the set of rules that the optimizer can use, it can potentially reduce the search space and as a result, reduce the optimization time sharply. Our experiments indicate that the essential set sizes for queries are typically small. If we had a priori knowledge of the essential set information of a query, we observe that we can obtain significant reductions in optimization time while *not* affecting the plan choice by the optimizer. This can be particularly useful for queries or stored procedures where the optimization time overheads are significant. Naturally, this raises the interesting question of

how we can predict a priori the set of rules that are likely to be essential for a query and opens up a novel research challenge. However, the significance of our study goes beyond just potentially speeding up the optimization step. In this paper, we also empirically study for the first time, the relationship between transformation rules and the complexity of plan diagrams [20] generated by the optimizer. Our results point to an opportunity to potentially reduce the complexity of plan diagrams (and hence improve stability of plans) by leveraging the knowledge of relevant rules for a workload. We also discuss the implications of these results for query optimizer testing. We present the results of our empirical study on a diverse set of query workloads (including real world and benchmark query workloads). To the best of our knowledge, this is the first work to present an empirical study focused on the effectiveness of transformation rules in a commercial query optimizer.

The outline of the paper is as follows. In Section II, we first review rule-based query optimizers, explain our experimental framework and describe the datasets and workloads used as part of our evaluation. Section III presents the results of identifying an essential set of rules for a query. Section IV presents the results of identifying the "most relevant" rules for a workload. In Section V, we look at the relationship between transformation rules and plan diagrams. We study the extent of reduction in optimization time of queries when the set of transformation rules is limited to the essential for the workload. (Section VI). We discuss the application of rule profiling to query optimizer testing in Section VII and present related work in Section VIII. We present our conclusions and discussions for future work in Section IX.

II. PRELIMINARIES

A. Overview of Rule Based Optimizers

In this paper we consider query optimizers that use a transformation rule based architecture as described in [10]. In this section we give a brief overview of a ruled-based framework for query optimization (a more detailed overview is available in [10]).

Transformation rule-based optimizers use a top-down approach to query optimization. The optimizer is initialized with a logical tree of relational operators corresponding to the input query. The goal of the optimizer is to transform the input logical tree to an efficient physical operator tree that can be used to implement the query. For this purpose, *transformation rules* are used to generate different alternative plans for executing a query. The set of rules that are available to the optimizer determines the search space of plans considered by the optimizer and thus is a key factor in determining the quality of the final plan.

There are two main kinds of transformation rules. *Logical transformation rules* transform logical operator trees into equivalent logical operator trees. Some examples of logical transformation rules include join commutativity and pushing group-by below join. *Implementation rules or physical transformation rules*, on the other hand transform logical operator trees into hybrid logical/physical trees. Example implementation rules include rules that transform a logical join operator into a physical hash join.

We note that other extensible optimizers such as Starburst [18] also leverage the idea of transformation rules during the query rewrite phase to generate alternative logical representations of the input query. In principle, the empirical methodology described in this paper can also be applied to such optimizers.

B. Notations

We use the following notations in the paper:

Set of transformation rules: We denote the set of transformation rules for the optimizer by $R = \{r_1, \ldots r_n\}$.

Execution plan and cost: For a given query q, we use $Plan(q)$, and $Cost(q)$ to refer to the execution plan chosen by the optimizer and its cost respectively.

We assume the ability to optimize a query when a subset of transformation rules is turned off. We note that many existing optimizers may already have support for this extension and the experiments in this paper can be performed in any database system that supports this functionality.

Let $S \subseteq R$ be a set of rules. We denote the execution plan and cost of a query q when only the rules in S are enabled (i.e. the remaining rules in R - S are turned off) by $Plan(q, S)$ and $Cost(q, S)$ respectively.

C. Experimental Setup

1) All the experiments were run on a machine with an AMD Opteron 2.4 GHz processor and 8 GB of main memory. We use Microsoft SQL Server as the database system and Windows Server 2003 as the operating system. The query workloads and datasets used for evaluation were the following.

1. **TPC-H**. We use the original workload of 22 queries and the original data generator [24] which generates data uniformly. In addition, we also use a modified version of the data generator which introduces skew in the data distributions of each column. We use a Zipfian distribution with a skew factor z=1. We use the 1GB version of the database. The TPC-H database (with uniform distribution) has indexes to enforce integrity constraints. The database with skew factor z=1 was tuned by creating additional indexes using the Database Tuning Advisor in Microsoft SQL Server [1].

2. **TPC-DS**. We use the TPC-DS benchmark (10 GB version) which is a new decision support benchmark in development [19]. This benchmark is intended to be the replacement of the existing TPC-H benchmark and includes uses a richer schema (that includes 25 tables) and a more comprehensive workload of 100 queries and uses skewed data distributions.

3. **SkyServer.** We use the personal edition of the SkyServer database (around 500 MB) along with a workload of 35

queries. These queries typically contain a small number of joins but make extensive use of user defined functions.

4. **InventoryDB**. We use a real world inventory database (around 15GB) of an online retailer. The workload is a set of 40 queries that are all instances of a single stored procedure which is a DSS style query that joins 8 tables and includes many aggregate functions.

5. **SalesDB**. We use a real world sales database (around 1 GB). The workload is a set of 35 reporting queries which join 6-8 tables.

III. ESSENTIAL SET OF RULES FOR A QUERY

The search space of a query optimizer is determined by the set of transformation rules used by it. In this section we first study how different transformation rules influence the plan choice of a particular query. In particular, we identify the set of rules that are sufficient to obtain the optimal plan for a given query. We refer to this as an essential set of rules for a particular query.

Definition: *Essential set of rules for a query*. Given a query *q*, an essential set of rules for query is a subset E of R such that Plan(Q, E) = Plan(Q, R) and this property is not true for any proper subset of E.

Figure 1 outlines the algorithm for computing the subset of rules that were "essential" for obtaining the optimal plan. We use a greedy algorithm similar to the Shrinking-Set algorithm which was proposed in [5] for the problem of computing an essential set of statistics to build for a query workload. The algorithm starts with the complete set of rules R. It considers each rule in turn. If the absence of the rule does not cause the plan to change, we deem the expression as "non-essential" and drop the rule. The running time of the algorithm is linear in the size of R. Note in general, multiple essential sets are possible.

1. $\mathcal{R}$ = Set of Rules in Optimizer
2. E = $\mathcal{R}$
3. **For** each r ε $\mathcal{R}$ **Do**
4. **If** Plan (Q, E – {r}) = Plan (Q, E)
5. E = E – {r}
6. **Return** E

Fig. 1 Essential Set of Rules for a Query

A. Results

The Figures 2-6 plot the sizes of the essential set of rules for all the queries in the different workloads considered in this paper. The Y-axis in the graphs denotes the number of rules in the essential set. The results for the TPC-H-uniform case were similar to the TPC-H-skewed database and are not shown.

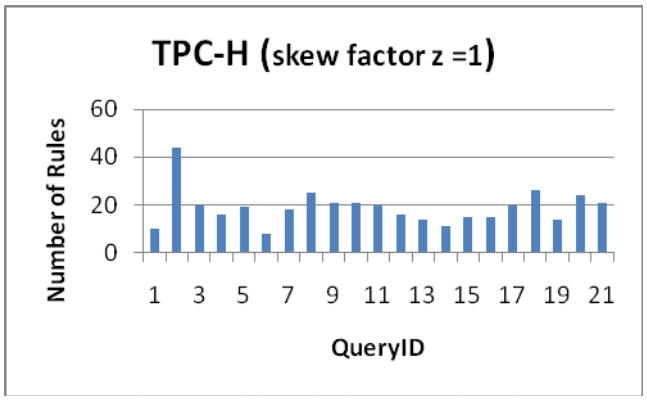

Fig 2. Essential Set Size for TPC-H (Skewed)

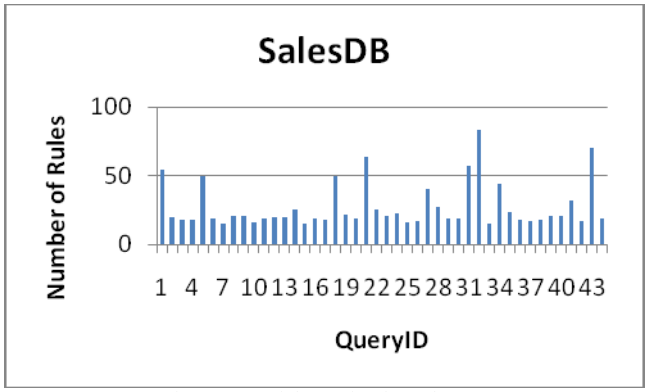

Fig 3. Essential Set Size for SalesDB

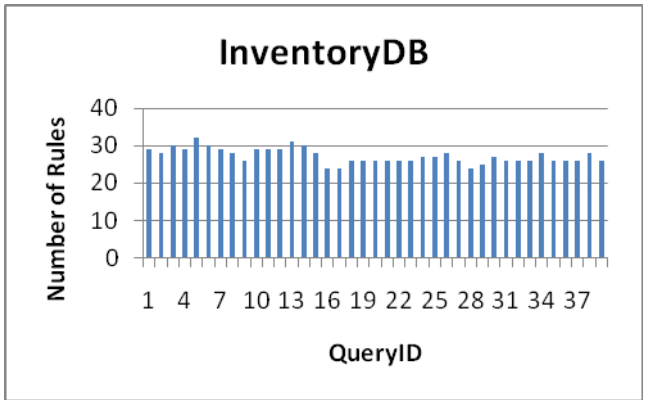

Fig 4. Essential Set Size for InventoryDB

The results indicate for a majority of the queries around 30 rules (around 10% of the entire set of around 350 rules) are sufficient to ensure that we get the optimal plan. There were a few queries where the number of rules in the essential set exceeded 60 (e.g., see Figure 6); these were some of the more complex queries in the TPC-DS workload that joins 11 tables and in addition uses the SQL WITH clause to handle multiple references to a nested sub-query. Of course, for any particular query, not all the rules will be syntactically relevant. For instance, transformation rules in SQL Server that are related to distributed query processing or XML processing will not be a part of an essential set for any query in these workloads. Even accounting for this (the rules that are relevant for SPJG queries are still a large fraction), the fact that the essential set

sizes are this small is interesting and we study how this information can be leveraged to reduce the optimization time for queries in Section VI. We also discuss implications of these results for query optimizer testing in Section VII.

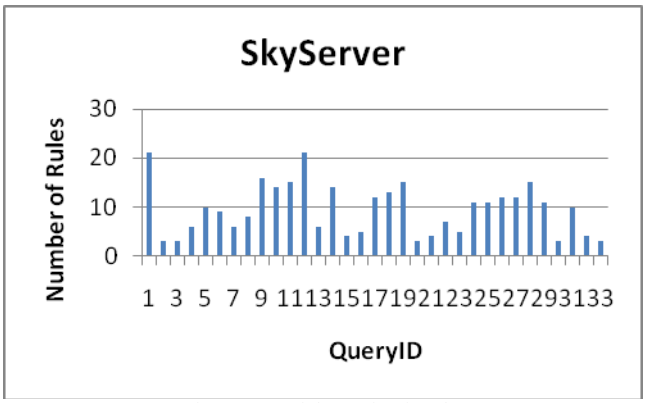

Fig 5. Essential Set Size for SkyServer

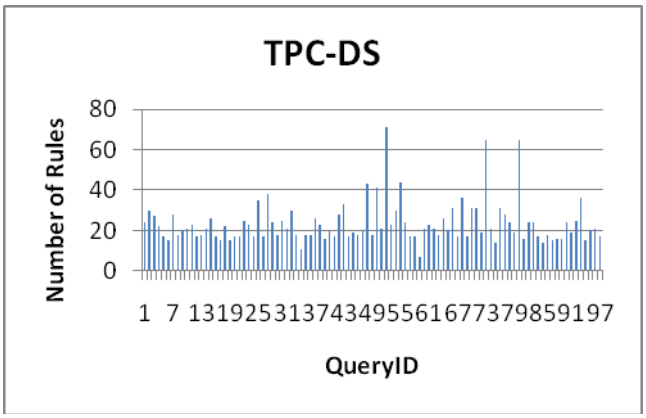

Fig 6. Essential Set Size for SkyServer

IV. IDENTIFYING RELEVANT RULES FOR A WORKLOAD

In the previous section, we analyzed which set of rules was sufficient to obtain the optimal plan for a particular query. In this section we extend this analysis to a workload, in order to understand how often a transformation rule impacts the choice of the optimal plan for a workload of queries. For instance, this information could be used to determine the most "important" rules for a workload of queries.

One way to profile rule relevance is to track how many alternative plans were generated during query optimization as a result of a particular rule firing. Note that while a transformation rule could have produced an alternate plan, it may still have not impacted the final plan chosen by the optimizer. For example, the alternate plan generated could be suboptimal and thus pruned by the optimizer. In this section, we present a metric to identify the most *relevant* rules for a workload of queries. We define a rule to be *relevant* for a query as follows.

Definition: *Relevant*(r_i, q): A rule r_i is defined to be relevant for a query q if *Plan*(q, {R}) != *Plan*(q, {R – r_i}) i.e., if optimizing the query with the rule turned off changes the plan chosen by the optimizer.

We define the relevance ratio for a rule r_i and workload of queries **W** (denoted as *RelevanceRatio*(r_i, **W**)) as the fraction of queries in **W** for which *Relevant*(r_i, q) is true.

We had defined the notion of an essential set of rules for a query in Section III. The relationship between the two metrics is as follows. In general, the union of the relevant rules for a query should be identical to its essential set. But this need not apply in some cases where the essential set of rules for a query may not be unique. In fact we observed a few degenerate cases where there were multiple essential sets for the same query. We can show that a rule is relevant for a query if and only if it is contained in every essential set of rules for a query.

In this section, for the different workloads we perform the following analysis. We compute the *RelevanceRatio* for each rule in R and for each workload we graph the *RelevanceRatio* in decreasing order for all rules that had non-zero *RelevanceRatios*.

A. Results

The results for the TPC-H database are shown in Figures 7 and 8. The total number of rules in the X axis indicates the number of distinct rules from R that was relevant for *at least one query* in the workload. As the graph shows this is fairly a small fraction of the entire set of rules. For example, for the TPC-H database only 70 rules out of the entire set of around 350 rules are relevant for at least one query. The graph also indicates the fact that the number of rules having a high *RelevanceRatio* is low. For instance, the number of rules that are relevant for 80% or more of the queries in TPC-H is less than 5.

Some of the most important rules for the TPC-H workload (having a relevance ratio of 50% or above), not surprisingly, include basic implementation algorithms such as sort-based algorithms for the OrderBy Operator and GroupBy operator, the hash join and sort merge algorithm for joins and a few implementation rules related to indexes. The most important logical transformations included rules for pushing down a selection over joins and the join commute rule and the rule for de-correlating a nested SELECT clause (e.g., for rewriting an EXISTS predicate to a join).

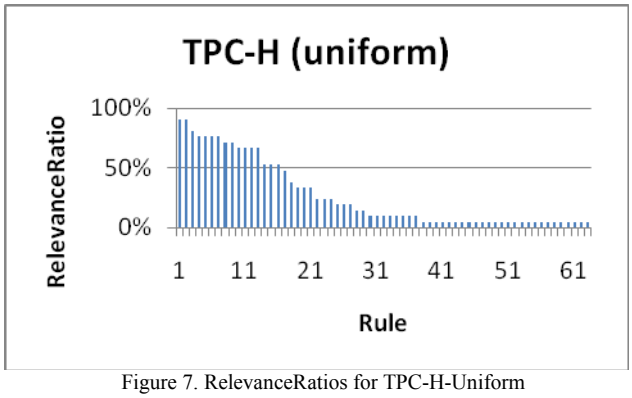

Figure 7. RelevanceRatios for TPC-H-Uniform

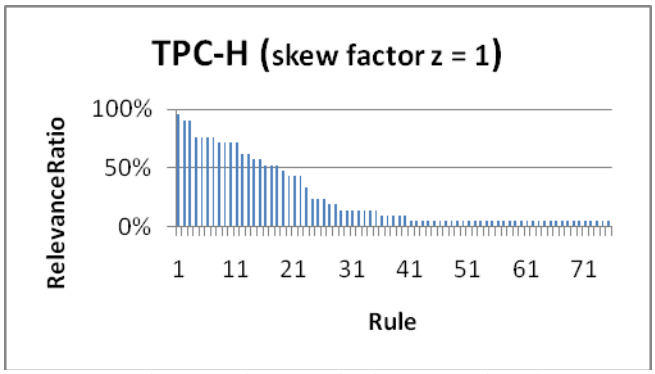

Figure 8. RelevanceRatios for TPC-H (Skewed)

Some of the well known transformation rules that had low relevance ratios (less than 5%) included rules for pulling up/pushing down a GroupBy operator and rules related to the outer join operator. For the skewed version of the TPC-H database, the important set of rules were largely similar but for one notable exception, we noticed an increased *RelevanceRatio* for rules pertaining to indexes; the skewed distribution (we use a skew factor z=1) increases the likelihood that some of the predicates are highly selective and thus making additional index rules more relevant (recall that this was a "tuned" database that also had a larger number of indexes).

The corresponding plots for the other workloads are shown in Figures 9 to 12 and the results are largely similar with the exception of the InventoryDB database.

The graph for the InventoryDB shows a larger number of rules having a high relevance ratio, for example half of the relevant rules have a relevance ratio of nearly 100%. This is largely because the queries in the workload are instances of the same stored procedure, thus there is a high degree of similarity in the queries and the corresponding relevant rules.

Among the set of workloads used in this paper, the TPC-DS benchmark has the largest number of relevant rules (around 130 rules). Interestingly, nearly all the rules that were relevant for the TPC-H database were also relevant for the TPC-DS database.

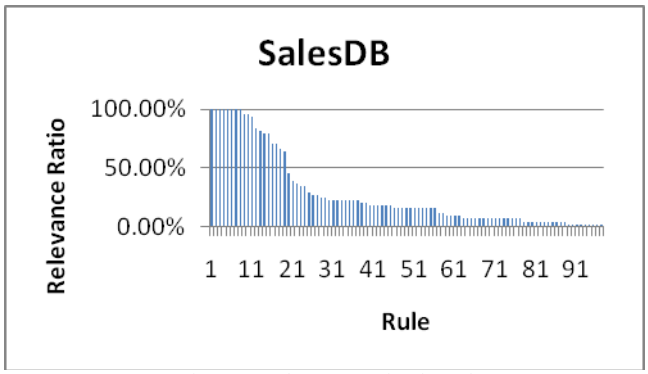

Figure 9. Relevance Ratios for Sales DB

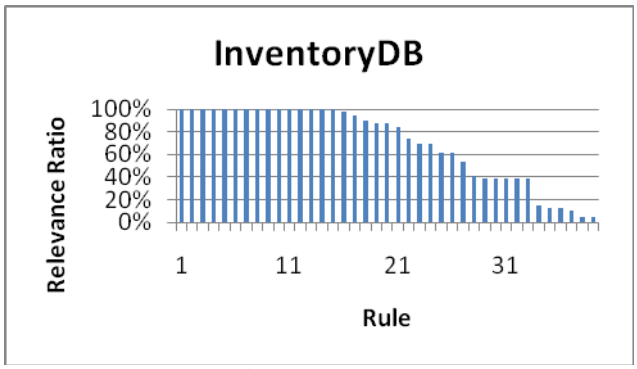

Figure 10. Relevance Ratios for InventoryDB

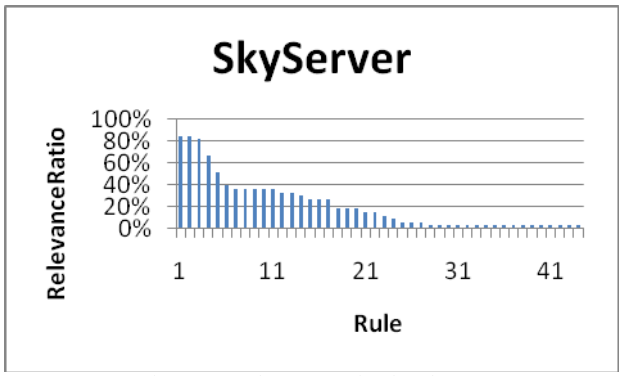

Figure 11. Relevance Ratios for SkyServer

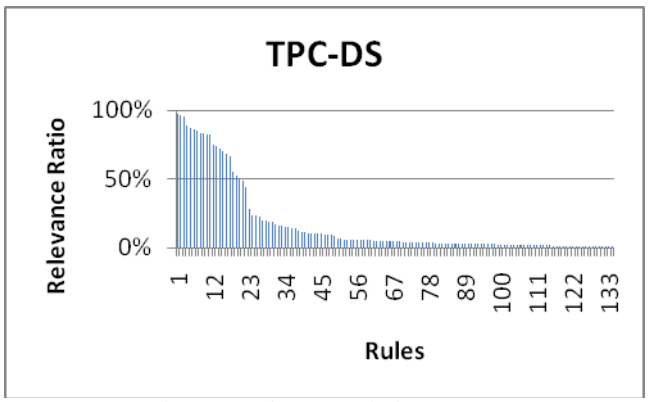

Figure 12. Relevance Ratio for TPC-DS

A common pattern among the results is the fact that there are relatively few rules (less than 20 in most cases) with a high *RelevanceRatio*. We show how we can leverage this information in Section V where we study the relationship between transformation rules and the complexity of plan diagrams generated by the optimizer. We also look at how the *RelevanceRatio* information can be used for optimizer testing in Section VII.

V. IMPACT ON PLAN DIAGRAM COMPLEXITY

The total number of rules used by an optimizer defines the search space of plans and thus directly contributes to the overall complexity of the optimizer. Plan diagrams [20] were introduced as a way of visually explaining the complexity of the plan space of a query optimizer. In this section, we study

the relationship between transformation rules and the complexity of plan diagrams for an optimizer.

(**Note to Readers:** We request viewing all the plots for this section directly from the PDF file or using a color printout in order to see the plan diagrams clearly)

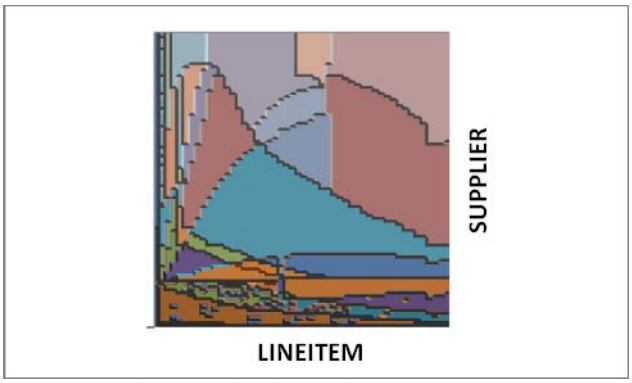

Figure 13. Plan Diagram for TPC-H Query 8

A plan diagram [20] is a pictorial enumeration of the plan choices made by an optimizer over the selectivity space. Figure 13 shows the plan diagram for TPC-H Query 8 (which is a query that joins 7 tables). We varied the selectivity space for the lineitem and supplier tables choosing 100 different selectivity values for each table (for a total of 10,000 data points). There are 54 distinct plans chosen by the optimizer in the entire space with some covering a very small fraction of the selectivity space. Figure 14 shows the corresponding cost diagram, which plots the optimizer estimated cost of these plans over the selectivity space.

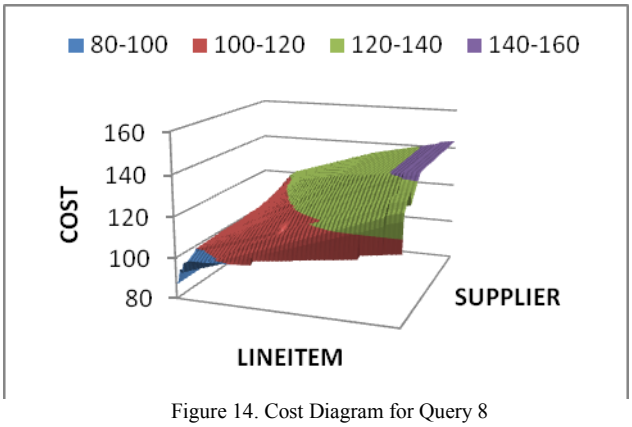

Figure 14. Cost Diagram for Query 8

By examining the plan diagrams of different queries, the authors in [20] argue that optimizer plan choices are perhaps, too fine grained. They propose the notion of a reduced plan diagram in which the optimizer produces a simpler plan diagram that is "close" enough in performance to the original. Plan diagram reduction as studied in [12][20] required the original plan diagram as input and left open the possibility of simplifying the optimizer at the "source", i.e. examining if the optimizer can be simplified so that they directly generate reduced plan diagrams.

Since the set of the transformation rules used in an optimizer directly influences the search space of plans considered by the optimizer, it is interesting to examine the relationship between the set of transformation rules used by the optimizer and the complexity of the resulting plan diagram. This has not been studied previously. We carried out the following experiment. We picked the 20 rules with the highest *RelevanceRatio* (see Figure 7) for the entire TPC-H workload. Intuitively, these are the 20 most "important" transformation rules for the TPC-H workload and as discussed in Section IV, these included implementation algorithms for group-by, order-by, joins as well as logical transformation rules for commuting joins and de-correlating nested select queries. We re-generated the plan diagram for Query 8 by turning off all the rules in the optimizer except these 20 rules. The corresponding plan diagram and cost diagram obtained are shown in Figure 15 and Figure 16. The number of distinct plans in Figure 15 is 14 (in contrast to 54 in the original plan diagram) and the maximum cost degradation at any point in the selectivity space when we compare the plan in the reduced diagram to the corresponding plan in the original plan diagram was 29.8% and the average degradation in cost (measured across the 10,000 data points) was only 6.6%.

We found a similar result for TPC-H Query 5 (which is a query that joins 6 tables). Figure 17 shows the original plan diagram for TPC-H Query 5 that has 33 distinct plans in the entire selectivity space. By utilizing only the top 20 rules with the highest *RelevanceRatio* for the TPC-H workload, we obtained the plan diagram in Figure 18 which has only 10 distinct plans. The average degradation at any point in the selectivity space when compared to the corresponding plan in the original cost diagram was 5.2% and the maximum degradation in cost was 26.1%.

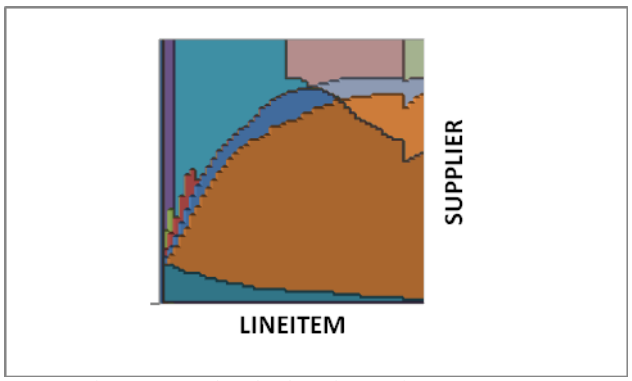

Figure 15. "Reduced" Plan Diagram for TPC-H Query 8

As these examples show, by carefully choosing the set of transformation rules to use, it is possible to obtain a much simpler plan diagram without sacrificing the quality of the plan significantly. These examples point to the opportunity of potentially reducing the complexity of plan diagrams by leveraging the relevance information of the rules for a workload. The appropriate rule selection for the plan reduction problem (as defined in [12]) for generating reduced plan diagrams is an interesting area of future work.

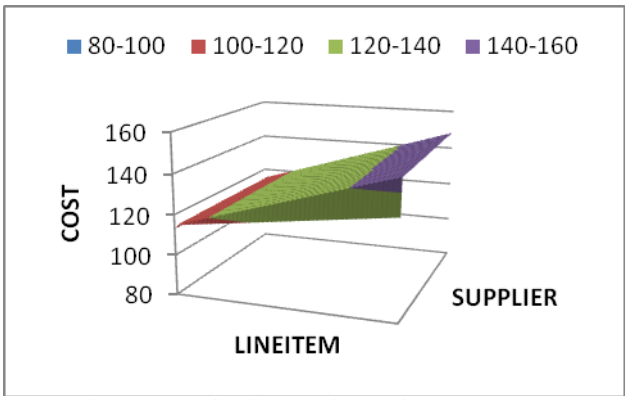

Figure 16. "Reduced" Cost Diagram for TPC-H Query 8

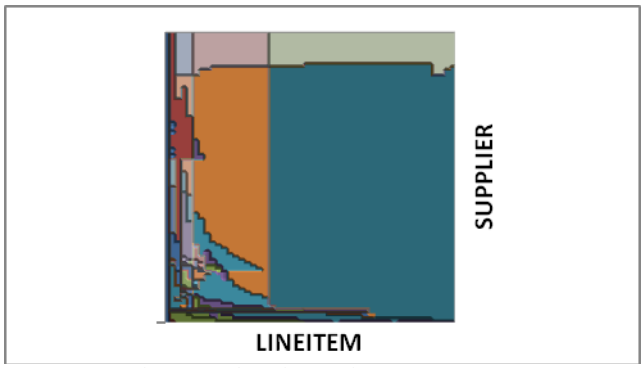

Figure 17. Plan Diagram for TPC-H Query 5

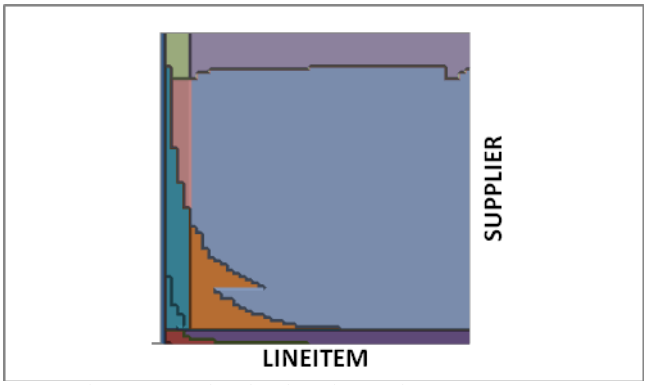

Figure 18. "Reduced " Plan Diagram for TPC-H Query 5.

VI. IMPACT ON OPTIMIZATION TIME

The search space of a query optimizer is largely determined by the set of transformation rules used by it. If we constrain the set of rules that the optimizer can use, it can potentially reduce the search space and as a result, reduce the optimization time sharply. In Section III, we observed that the essential set of rules for a large fraction of the queries is typically small. In this section, we examine how this impacts the time taken to optimize the query.

We optimized each query by turning on only the rules in its essential set and turning off all other rules. Note that this would guarantee the same optimal plan. We plot the reduction in optimization time for the workloads where the optimization time was significant, we do not show the results for the workloads where the optimization times were negligible (e.g. the SkyServer database).

Figures 19-21 show the reduction in optimization times for the TPC-H databases and for InventoryDB.

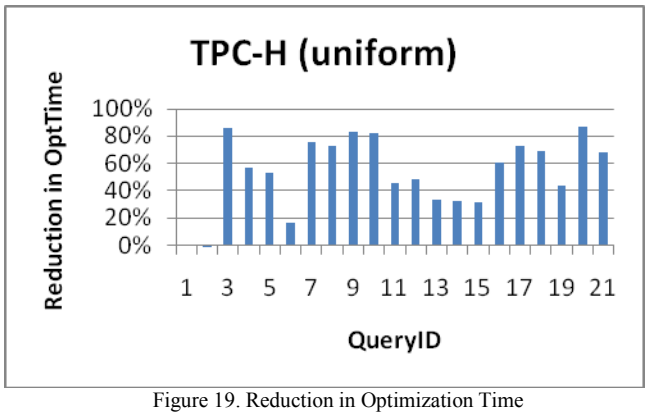

Figure 19. Reduction in Optimization Time

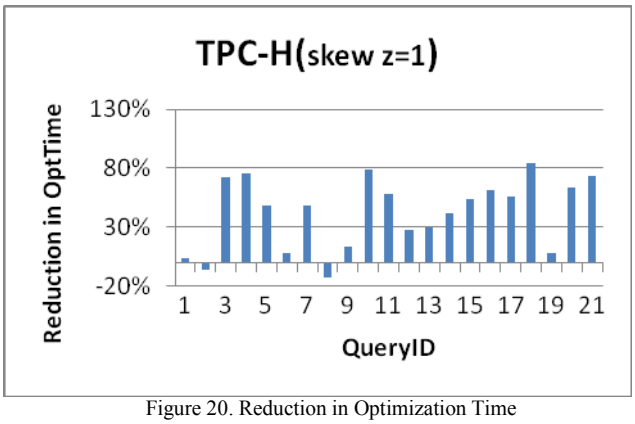

Figure 20. Reduction in Optimization Time

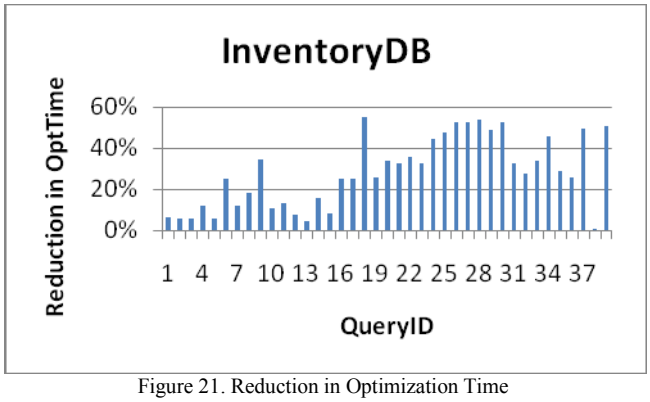

Figure 21. Reduction in Optimization Time

The results indicate that the savings in optimization time can be substantial (e.g. 50% or more in several cases). Note that this reduction in optimization time comes *does not affect* the quality of the plan; by including an essential set of rules (see Section III) for a query, we guarantee that we find the same optimal plan.

The results for the TPC-DS were largely similar (and not shown). Interestingly, there were a few queries where the optimization times were higher than the other workloads (10's

of seconds) and the corresponding reduction in optimization time was also much higher (around 90% reduction).

Since the essential set sizes are typically small and can potentially result in significant savings in optimization time, one can consider adding *rule hints* to the repertoire of query optimizer hints. The idea is to provide a hint to the query optimizer to use a smaller set of rules for optimizing a query. Such a hinting mechanism could be especially important for queries or stored procedures where the optimization time overheads are non-trivial. Profiling the set of rules that actually influence the plan choice for a particular query provides a useful way to derive appropriate rule hints for a query.

It is also interesting to examine if we can predict the essential set of rules of a query before query optimization, perhaps by leveraging the essential set information of "similar" queries and applying machine learning techniques. Such information would be useful for a meta-optimizer (e.g., as described in [13]) which can tune the optimizer appropriately for each input query.

We also noticed a few cases where the optimization time increased when we constrained the number of rules (e.g., see Figure 20); these point to interactions between the pruning strategies/time out mechanisms in the optimizer and the transformation rules. We revisit this interaction in Section VII where we discuss the application of rule profiling to optimizer testing.

VII. IMPLICATIONS FOR OPTIMIZER TESTING

Apart from giving feedback on which transformation rules are useful in real workloads, transformation rule profiling has important implications for query optimizer testing (see [9][22] for an overview of issues related to query optimizer testing). In this section, we discuss a few applications of rule profiling to query optimizer testing.

Tracking Interaction between Rules and Pruning: There is a fundamental tradeoff between optimization time and the quality of the plan output by the optimizer. In order to enable optimizers to produce plans "quickly" irrespective of the complexity of the input queries, modern optimizers use a variety of pruning strategies/timeouts etc. These can at times interact with transformation rules in non-intuitive ways. For example, we presented ways to identify relevant rules for a workload of queries (Section IV). One would expect the cost of the plan generated by the optimizer only to increase if we were to turn off a rule that is relevant for it. However, we found instances where turning off a relevant rule can sometimes result in a better plan. Though in many of the cases, the cost difference between the plans was negligible, we did find examples where the difference in estimated cost was 30% or more (e.g., for around 1% of the cases for the TPC-DS database). Detecting such cases can help query optimizer testers to identify potential interactions between transformation rules and the heuristics used for pruning the search.

Measuring Coverage of Test Suites: Tracking the relevant rules for a workload (Section IV) can be important for analyzing the test suite coverage for query optimizers. For instance, consider Figure 22 which plots the *RelevanceRatios* for the union of all the workloads used in this paper. Recall that the number of distinct rules in this plot indicates the total number of rules that are relevant for at least one query in the entire workload. In this case, even after using a diverse set of workloads, only around 40% of the rules were really relevant. Of course, as mentioned in Section III, some of the original rules are not syntactically relevant for these workloads (for example, transformation rules in SQL Server that are related to the CUBE operator in SQL). But tracking this metric for the test suites could be potentially important for the following reasons.

Query optimizers are typically tested using different workloads that include workload traces from real customer applications, and other synthetic benchmarks. By tracking the relevance ratios we can understand how well our current test suites cover the optimization rules and if they need to be augmented with additional queries. If some rules are still not covered in this plot, then it could potentially indicate a bug in the code that fires the corresponding rule or it could indicate the fact that a particular rule is too specialized to be useful in real workloads.

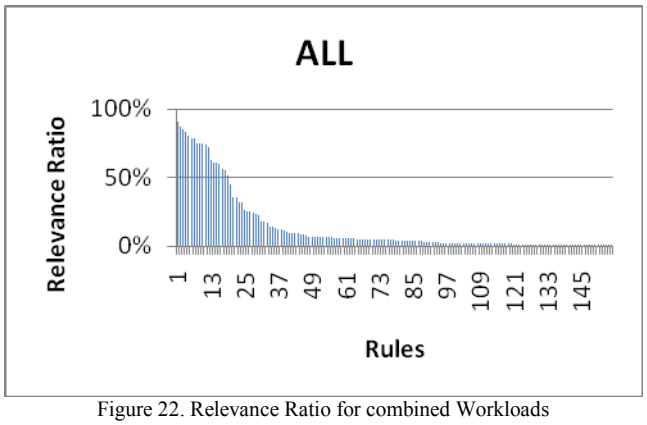

Figure 22. Relevance Ratio for combined Workloads

Benchmarking Search Strategies: Tracking the essential set of rules for a query can be very useful for benchmarking the current search strategies used in an optimizer. The time taken to optimize the query using an essential set of rules provides a *lower bound* (for the current query optimizer) on how quickly we can compute the optimal plan for a particular query. By comparing the current optimization times to this lower bound, we can evaluate the effectiveness of the current search strategy used by the optimizer. For instance, Figure 19 shows the reduction in optimization time for the TPC-H workload that can be achieved by using the essential set information for each individual query. For example, for some queries, the reduction in optimization time is less than 20%, indicating that the current search strategy is already very effective in pruning the irrelevant rules for these queries.

VIII. RELATED WORK

There has been extensive work on how to architect query optimizers in order to make them extensible. Many modern optimizers use a transformation rule-based framework including industrial query optimizers such as IBM Starbust [18], Microsoft SQL Server [10], Oracle[2], Tandem's NonStopSQL [3] as well as academic prototypes such as the Volcano optimizer [11].

There has also been work centered on adding new transformation rules to the optimizer (see [4] for an overview). Some of the important classes of rules studied thus far include rules for pulling-up/pushing down a group-by operator (e.g., [6][23]), handling nested sub-queries (e.g., [15][17]) and handling outer-joins (e.g., [8]).

The idea of plan diagrams was introduced in [20] as a way of visually explaining the complexity of the execution plan choices of a query optimizer. Techniques for producing a "reduced" plan diagram are presented in [12]; these algorithms required the original plan diagram as input and left open the possibility of examining if the optimizer can be simplified so that they directly generate reduced plan diagrams. In this paper, we showed that by leveraging the relevance information of rules for a workload, it is possible to generate simpler plan diagrams without significantly impacting the quality of the plan.

In spite of extensive work on architecting query optimizers, there has been little work focused on empirical studies centered on the query optimizer. One of earliest papers in this space was [16] which validated the cost model of the R* query optimizer. The work in [14] presented analytical and empirical results to better understand the tradeoffs in the search space used by the query optimizer (in particular left-deep trees vs. bushy trees). To the best of our knowledge, this is the first work to present an empirical study focused on how transformation rules are utilized in a query optimizer.

IX. CONCLUSIONS

In this paper we present the results of an empirical study focused on evaluating the effectiveness of transformation rules using Microsoft SQL Server query optimizer and a diverse set of real world and benchmark query workloads.

We presented techniques to track two related metrics: 1) the essential set of rules for a query and 2) the most relevant rules for a workload. We empirically observed that the essential set sizes of most queries were small and that there are relatively few rules with high *RelevanceRatios*. We discussed how this information can be potentially leveraged for simplifying plan diagrams, reducing optimization time and testing query optimizers.

As part of future work we intend to further extend the rule profiling techniques to cover other aspects of transformation rules such as tracking rule dependencies. Of course, transformation rules are just one component in a query optimizer which also includes other important modules such as cardinality estimation etc. Building a set of analysis tools that is centered on the different modules of the query optimizer is an interesting direction for future work.

REFERENCES

[1] S.Agrawal et al. Database Tuning Advisor for Microsoft SQL Server 2005. In Proceedings of VLDB 2004.

[2] R.Ahmed et al. Cost Based Query Transformation in Oracle. In Proceedings of VLDB 2006.

[3] P.Celis. The Query Optimizer in Tandem's new ServerWare SQL Product. In Proceeding of VLDB 1996.

[4] S. Chaudhuri. An Overview of Query Optimization in Relational Systems. In Proceedings of PODS 1998.

[5] S.Chaudhuri, V.Narasayya. Automating Statistics Management for Query Optimizers. In Proceedings of ICDE 2000.

[6] S.Chaudhuri. K.Shim. Including Group-By in Query Optimization. In Proceedings of VLDB 1994.

[7] U.Dayal. Of Trees and Nests: An unified approach to processing queries that contain nested subqueries, aggregates and quantifiers. In Proceedings of VLDB 1987.

[8] C.Galindo-Legaria, A.Rosenthal. How to Extend a Conventional Optimizer to Handle One and Two-Sided OuterJoin. In Proceedings of ICDE 1992.

[9] L.Giakoumakis, C. Galindo-Legaria. Testing SQL Server's Query Optimizer: Challenges, Techniques and Experiences. IEEE Data Engineering Bulletin 2008 vol. 31 (1).

[10] G.Graefe. The Cascades Framework for Query Optimization. Data Engineering Bulletin, 18(3), 1995.

[11] G.Graefe, W.McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In Proceedings of ICDE 1993.

[12] D.Harish, P.Darera, J.Haritsa. On the Production of Anorexic Plan Diagrams. In Proceedings of VLDB 2007.

[13] I.Ilyas et al. Estimating Compilation Time of a Query Optimizer. In Proceedings of ACM SIGMOD 2003.

[14] Y.Ioannidis, Y.C.Kang. Left Deep vs. Bushy Trees: An Analysis of Strategy Spaces and Implications for Query Optimization. In Proceedings of ACM SIGMOD 1991.

[15] W.Kim. On Optimizing a SQL-like Nested Query. ACM TODS Vol. 9 No. 3, 1982.

[16] L.Mackert, G.Lohman. R* Optimizer Validation and Performance Evaluation for Local Queries. In Proceedings of ACM SIGMOD 1986.

[17] M.Muralikrishna. Improved Unnesting Algorithms for Join Aggregate SQL Queries. In Proceedings of VLDB 1992.

[18] H.Pirahesh, J.Hellerstein, W.Hasan. Extensible Rule Based Query Rewrite Optimization in Starburst. In Proceedings of ACM SIGMOD 1992.

[19] M.Poess, R.O.Nambiar, D.Walrath. "Why you should run TPC-DS: A Workload Analysis". In Proceedings of VLDB 2007.

[20] N.Reddy, J.Haritsa. Analyzing Plan Diagrams of Query Optimizers. Proceedings of VLDB 2005.

[21] Praveen Seshadri, Hamid Pirahesh, T. Y. Cliff Leung: Complex Query Decorrelation. ICDE 1996: 450-458

[22] M.Stillger, J.C.Freytag. Testing the quality of a query optimizer. Data Engineering Bulletin, 18(3), 1995.

[23] W.Yan, P.Larson. Eager Aggregation and Lazy Aggregation. In Proceedings of VLDB 1995.

[24] TPC-H Benchmark. http://www.tpc.org