Plan Space Analysis: An Early Warning System to Detect Plan Regressions in Cost-based Optimizers

Florian M. Waas
EMC^2 Corp.
florian.waas@emc.com

Leo Giakoumakis
Microsoft Corp.
leogia@microsoft.com

Shin Zhang
Microsoft Corp.
shin.zhang@microsoft.com

ABSTRACT

Plan regressions pose a significant problem in commercial database systems: Seemingly innocuous changes to a query optimizer component such as the cost model or the search strategy in order to enhance optimization results may result in unexpected and detrimental changes to previously satisfactory query plans.

Database vendors spend substantial resources on quality assurance to guard against this very issue, yet, testing for plan regressions in optimizers has proven hard and inconclusive. This is due to the nature of the problem: the optimizer chooses a single plan—*Best Plan Found* (BPF)—from a search space of literally up to hundreds of millions of different plan alternatives. It is standard practice to use a known good BPF and test for changes to this plan, i.e., ensure that no changes have occurred. However, in the vast majority of cases the BPF is not be affected by a code-level change, even though the change is known to affect many plans in the search space.

In this paper, we propose a holistic approach to address this issue. Instead of focusing on test suites consisting of BPFs we take the entire search space into account. We introduce a metric to assess the optimizer's accuracy across the entire search space.

We present preliminary results using a commercial database system, demonstrate the usefulness of our methodology with a standard benchmark, and illustrate how to build such an early warning system.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing*

General Terms

Experimentation, Measurements, Performance

DBTest '11 June 13, 2011, Athens, Greece

1. INTRODUCTION

Query optimization is at the heart of a relational database system. The optimizer compiles a declarative query description, e.g., SQL, XQuery, MapReduce, etc. into an executable *query plan*. A query plan is the result of a non-trivial process that takes into account a large number of factors, such as the anticipated cost of a plan fragment, and arrives at a decision which plan most likely offers the best performance[1]. This plan is called the *best plan found*, short BPF. The BPF is not necessarily the *optimal plan*. Rather, the BPF is a compromise of modeling the anticipated execution at a level as detailed as possible, yet, spending no more resources and time on the problem as is deemed necessary. The BPF is chosen based on an model that approximates the actual execution environment.

1.1 Plan Regressions

Plan regressions are changes in query plans that lead to inferior performance.

Plan regression may occur in many different situations. The two most common ones are:

- Code-level changes to the optimizer trigger a bad plan choice, e.g., attempts to improve the search strategy, more accurate cardinality estimation, more sophisticated cost model, or the mere correction of software defects;

- Changes to the optimizer's input parameters trigger a bad plan choice, e.g., minor changes to the logical or physical design of the database; changes to the underlying data through additional data loads, or simply recomputed statistics;

The effect of plan regressions can be frustrating for users and strategies for mitigating them in order to buffer users from their effect have been proposed, see e.g. [1]. These strategies consist of techniques that carefully vet new plans and only deploy them after they have been proven better than the previous, establishing monotonic performance behavior.

While such techniques address customers' concern and are highly valuable for the end-user, they do not help engineers to direct their efforts to actually improve the product.

Therefore, we consider plan regressions independent of their possible impact on customers but primarily on their

[1]Other optimization goals such as robustness, resource consumption etc. are conceivable. For simplicity we use performance as the optimization goal. Our methods apply to other objective functions as well.

effect on the development process without which a product cannot be improved.

In particular, we are interested in regressions pertaining to the appropriate assessment of plans, i.e., whether the better of two alternatives is chosen.

1.2 Common Test Practice

Quality assurance for optimizers has a long tradition in database development and seen heightened interested recently, see e.g., [7, 2, 3].

The problem of plan regressions has plagued database vendors for a long time. Despite huge efforts and substantial costs to most development organizations the issue has not been addressed satisfactorily.

The standard test practice is as follows:

1. Define a plan regression test suite; chosen for their relevance, often including benchmark queries, customer workloads, etc.
2. Retrieve the BPF for each query in a form that can be stored in the test database or in flat files;
3. Execute the test suite, retrieve BPF using the new configuration and compare against the previously stored one;
4. In case of discrepancy, identify the code-level change that triggered the plan change, engage developer in discussion to understand if the detected plan change is beneficial or needs to be corrected;
5. If code change is accepted, update plan regression suite;

The crux of the problem are plan regressions that are considered necessary or valid. The most prominent case is the classic two-wrongs-make-a-right: Occasionally, bad estimates cancel each other out. By correcting one of two estimates the remaining inaccuracy may lead to a regression. Yet, correcting the root cause of the bad first estimate might be considered a general improvement to the overall optimizer quality and should therefore be accepted despite causing a regression. These decision have to be made on a case-by-case basis.

There are several major problems with this practice. First, only changes to the BPF are tested and potentially far-reaching changes that do not affect the BPFs of the regression suite go undetected. Increasing the number of tests in the suite may increase the coverage. However, given the massive numbers of plan alternatives modern optimizers consider for even modestly complex queries, increasing the test suite in the hopes that BPFs are distributed suitable is a loosing proposition. Secondly, the process as outlined above requires significant manual intervention each time a plan change is detected. Understanding the cause for the change and determining if the change should be accepted as the new reference plan is an involved process. As a result such plan regression suites are naturally confined to small sizes which is at odds with the previous requirement of collecting as many data points as possible.

2. DESIDERATA

The following is a list of desiderata for a measure or mechanism to detect plan regressions of an optimizer early and conclusive that we collected from numerous discussions with practitioners and database implementers and researchers as well as own research [5, 8, 4].

As mentioned above, we are primarily looking at plan regressions from a developer's point of view, with the intent to improve the optimizer. That is, we are not only interested in preventing regressions in the form of bad plan choices but would also like to detect improvements the same way.

Simple. The measure should be a *simple* value. Multi-dimensional measures—i.e., vectors of individual (often even unrelated) measures—are difficult to compare.

Transparent. The measure should express a strong and semantically clear assessment. Ideally, the measure relates to a simple and tangible concept of goodness, e.g., bad/good plan choices the optimizer has made during a single test.

Agnostic. The measure should be agnostic of the particular optimizer technology used. Specifically it should not attempt to reverse engineer components and therefore become overly dependent on the current implementation because future changes may significantly alter or even remove these components.

Targeted. A query optimizer tries to generate the optimal plan for a given target platform, i.e., a combination of execution and storage components. Over the course of several releases, the underlying execution engine may change substantially. The measure must take into account that each version of the optimizer may optimize for a different target platform.

Surgical. The measure should apply to individual queries. This will enable implementers to troubleshoot specific optimizer problems that are otherwise buried in larger workloads or hidden by interactions of other components.

Specific. The measure, or an extension of it, should enable application-specific comparisons. That is, it should apply to *any* workload in addition to individual queries. This will enable test engineers to assess plan regressions for a given application scenario. Relying on a single benchmark is generally considered insufficient.

Practical. The measure should be straight-forward to compute and lend itself easily to automation.

The desiderata as listed above capture the most important aspects of measuring an optimizer's quality in general. Consequently, we believe a mechanism that satisfies most or all of these may be also an important building block for a more general optimizer benchmark.

3. PLAN SPACES

A *plan space* S_Q is the set of alternative plans that an optimizer considers for an individual given query Q. Each element of the plan space is a complete query plan that, when executed, produces a valid answer to the given query. The plan space is specific to the implementation of an optimizer in that it is *not* the set of plans over hypothetical relational algebra but strictly the set of plans actually *considered* by the optimizer on hand.

A plan space is invariant with respect to possible changes in data—just like a query is typically run over data that changes as part of the normal production workflow. We refer to entirety of parameters that describe the data as *database configuration* D.

Two plans within a plan space are of particular interest: the *optimal plan*, and the BPF, i.e., the plan the optimizer considers the optimal plan. In practice, the BPF is the optimal plan only for trivial queries with few alternatives or trivial database configuration with little room for estimation errors.

The following observation will be critical for our methodology:

Observation. *Given a plan space S_Q for a query Q, for any two plans $p_1, p_2 \in S_Q$ there exist two database configurations D_1 and D_2 such that p_1 is optimal for D_1 and p_2 is optimal for D_2. In general, p_1 will not be optimal in D_2 and p_2 not optimal in D_1.* □

In other words, for any plan of a given plan space, we can carefully manufacture a database configuration for which this plan will be optimal, i.e., it will not be outperformed by any other plan in the plan space.

Example. For example, consider the case of a simple join between ORDERS and CUSTOMERS of the TPC-H schema, with a non-covering index on C_CUSTKEY, the primary key of Customers:

```
SELECT *
FROM CUSTOMERS, ORDERS
WHERE O_CUSTKEY = C_CUSTKEY
AND O_ORDERDATE >= date '2010-01-01'
O_ORDERDATE < date '2010-11-01'
ORDER BY O_orderkey
```

Depending on the data and selectivities of the predicates the optimal plan could contain an index lookup join into CUSTOMERS, a index scan with a sort-merge or a hash join following. For some database configurations the index will not play a role at all since the fetching of the remaining rows might be too costly. In addition, a number of additional optimizations and plan variants may apply.

The optimizer we experimented with considers a total of 103 different plans for this rather trivial query. That means, there are up to 103 database configurations that will lead to 103 different optimal plans. □

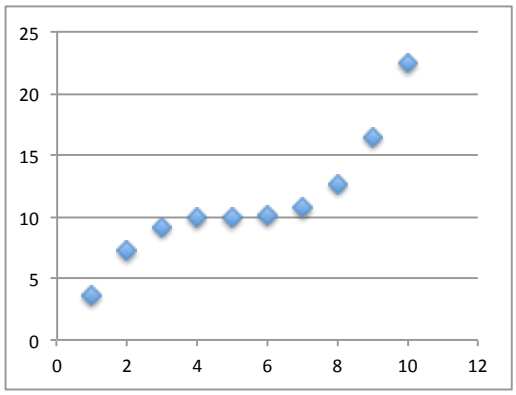
It is important to note that the concept of a plan space is independent of a cost model. Even a purely heuristic rule-based optimizer that does not deploy any costing at all operates over a plan space. Its decision may not be driven by a cost model but it still is able to generate different plan alternatives depending on the database configuration.

4. PLAN SPACE ANALYSIS

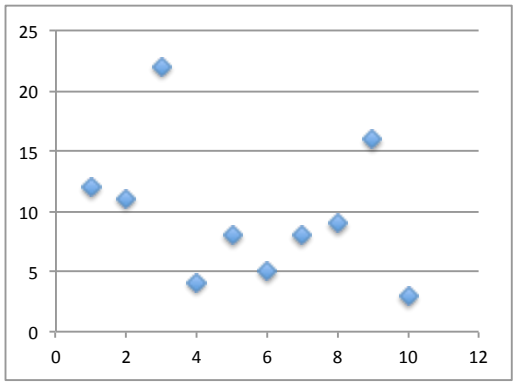
In this section we will develop a measurement that determines how well an optimizer models a query's search space.

4.1 Motivation

Let us assume for the moment a *perfect* optimizer. A perfect optimizer is one that is able to establish a partial order on all plan alternatives according to their actual performance. That is, for each pair $(p_1, p_2) \in P_Q$ this optimizer is able to decide $p_1 \leq p_2$ if p_1 outperforms p_2 and, conversely, $p_2 < p_1$ if p_2 outperforms p_1. A perfect optimizer is therefore able to correlate estimated and actual execution cost monotonically as indicated with the example in Figure 1a. The X-axis represents estimated costs, the Y-axis actual execution time. The correlation function does not need to be linear. Actually, in practice, we often find alternative plans have been differentiated by the optimizer correctly in terms of which plan outperforms the other although the exact difference was over or under estimated compared to other distances between pairs of plans. Most notably, the BPF at (1, 3.6) corresponds to the optimal plan.



(a) Monotonic correlation



(b) Non-monotonic correlation

Figure 1: *Schematic scatter plots*

On the other hand, and in our experience much more realistic, is the case of a less-than-perfect optimizer as illustrated in Figure 1b. The correlation between estimated and actual is much weaker. And, importantly, the BPF at (1, 12) is more expensive than 70% of the other plans, although no worse than three times the optimal. This second plot indicates the optimizer did a mediocre job at best.

Most optimizer test suites try to determine whether an optimizer is able to distinguish the BPF from any inferior plan, in the hopes that this implies that the BPF is optimal. However, based on the observation in the previous section that for every plan exists a database configuration for which this plan is optimal we expect that a high quality optimizer be able to order any two plans correctly, regardless of whether

one of them is the optimal plan or the BPF.

Intuitively, the better an order within the plan space an optimizer is able to establish, the better this optimizer's overall quality. Hence, we will use the ability to order plans correctly as a measure for the accuracy of the optimizer's cost model, in the following.

Using this insight, we express the correlation in a plan space as a function of the number of incorrect orderings that we detect. This measure will allow us to detect detrimental plan changes, independent of whether or not the BPF is affect by the change.

4.2 Definition

Formally, we define the correlation of a plan space as follows:

Given a query Q and a set of alternative plans $p_1, \ldots, p_n$, let X_i denote the rank of plan p_i according to the optimizer and Y_i denote the rank of plan p_i according to actual execution time. We define ρ as Spearman's Rank Correlation Coefficient *of X and Y:*

$$\rho(Q, n) = 1 - \frac{6}{n(n^2-1)} \sum_{i=1}^{n} (X_i - Y_i)^2$$

For a workload W consisting of queries $Q_1, \ldots, Q_m$ we define ρ as

$$\rho(W, n) = \frac{1}{m} \sum_{i=1}^{m} \rho(Q_i, n)$$

□

In statistics, a number of different measures for correlation are used. We chose to use the Spearman's Rank Correlation based on its highly suitable properties. In particular, it expresses the statistical dependence between two variables by assessing how well the relationship between the two can be approximated with a monotonic function. The Spearman Rank Correlation, and hence our measure ρ, produces values between -1 and 1, with 1 denoting perfect correlation between estimated and actual ranking. For a detailed discussion of Spearman's Rank Correlation Coefficient, see e.g. [6].

We introduced n as a variable for practical purposes. We maintain a constant n across a workload although queries may vastly differ in size of their respective search spaces.

Example. In Figure 2, a sample of 11 plans—10 randomly chosen plans plus the BPF—with their estimated and actual execution costs is listed. For the purpose of this example, the actual query parameter are irrelevant. Columns X and Y show the ranking of the plans according to their estimated and actual cost.

Figure 3 is a scatter plot of the ranks. It clearly shows 2 "mistakes" the optimizer made as outliers from an otherwise perfectly correlated plot. The plans with actual ranks 3 and 11 were incorrectly ranked 5 and 8. The resulting $\rho(Q, 11)$ score is 0.918. One might argue that the first mistake is less severe than the latter and therefore should contribute differently to the score. However, we decided to stick with the simpler rank correlation independent of additional data points. We will discuss possible extensions to the method to address potential issues like this in the next sections though. □

plan	estimated	actual	Y	X
1	1.000	1.000	1	1
2	1.004	1.106	2	2
3	2.337	1.178	5	3
4	2.165	1.454	3	4
5	2.287	1.456	4	5
6	2.393	1.465	6	6
7	5.418	1.938	7	7
8	5.750	1.944	9	8
9	5.757	1.946	10	9
10	5.828	1.946	11	10
11	5.536	1.947	8	11

Figure 2: *Normalized estimated and actual execution costs of a sample of 10 plans for Q5 of TPC-H; ordered by actual execution cost; (X = rank as per actual, Y = rank as per estimate)*

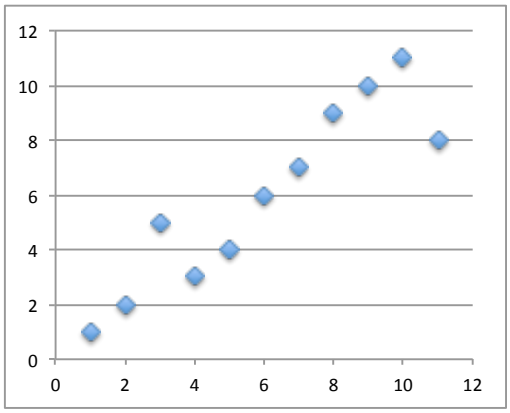


Figure 3: *Scatter plot of actual vs. estimated ranks. 2 plans (rank 3, rank 11) out of order;*

ρ meets the desiderata as given in Section 2: it condenses the optimizer's decisions into a simple yet transparent measure, is agnostic of the underlying technology and captures that the optimizer produces a plan for a very specific target platform.

4.3 Practical Considerations

While experimenting, we came across a few practical issues worth noting, mainly regarding the choice of the underlying sample of plan alternatives.

First, to the best of our knowledge only one commercial optimizer type offers the ability to sample uniformly from a plan space. However, neither uniformity nor full support for sampling will be needed to determine ρ for a query or a workload. Actually, many database administrators perform the simplest of assessments of ρ on a daily basis whenever they suspect a BPF is noticeably supoptimal: using special controls such as hints or other tuning controls they force a number of plan alternatives execute them and the compare them to the BPF. In a way, they generate samples of plans manually.

Second, including plans that are significantly more expensive than the BPF in the plan space analysis poses additional difficulties: these plans may take exceedingly long to execute

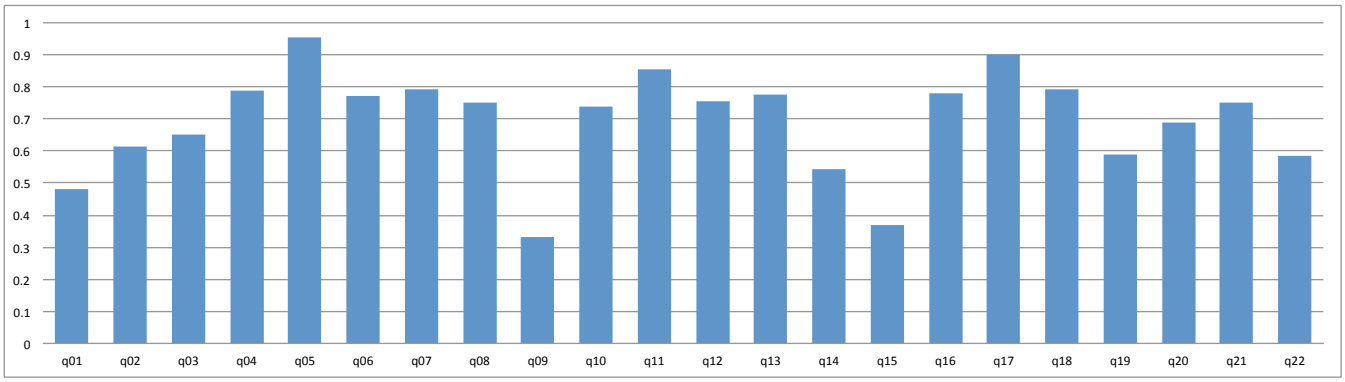


Figure 4: *Correlation coefficients of TPC-H queries; higher values indicate higher correlation between estimated and actual costs across the query's entire plan space*

and render test runs impossibly long. For practical reasons we exclude plans that have been estimated at more than 10 times the cost of the BPF from the sample as detailed above.

Third, it is important for the accuracy of ρ to be able to differentiate plans with similar execution cost conclusively. This may be difficult if plans have almost identical execution time. We have excluded plans whose execution is too close to another member of the sample.

Lastly, we suggest using values of n of at least 10 or greater to achieve stable repeatable results. Also, queries that have less than 10 alternative plans should be excluded from the rating/workload.

5. EXPERIMENTS

In order to validate the basic ideas behind plan space analysis we present some initial experiments.

5.1 Setup

We chose TPC-H as a test bed for the high degree of readers' familiarity. We use a 10GB database size although the actual size is not relevant for our purposes.

For any given query we select n random plans using the sampling mechanisms of [8]. We executed each plan at least twice and took the average execution time.

We also included the BPF as a baseline. For practical reasons, we focused only on plans that run within a small multiple of the time of the optimal plan. This mechanism is necessary as a random sample of plans may contain arbitrarily bad plans too.

5.2 Workload

In Figure 4, the correlation coefficient for all queries of the TPC-H suite is shown. The plot shows overall very good ρ scores for most queries—but identified also several problem queries, which are currently under investigation by the development team as a consequence of our experiments.

Given the TPC-H data set, it may seem surprising at first that our optimizer did not achieve a perfect score. Counter to common believe, most commercial optimizers are actually biased against TPC-H—and that for good reasons: customer data contains almost always correlations that go undetected. Most optimizers try to anticipate this and do not treat multiple predicates fully independent, among other things. This type of bias has proven valuable in a wide variety of application scenarios. However, in the case of TPC-H, the data is unnaturally independent and, hence, the bias is slightly disadvantageous for this query set.

5.3 Plan Regressions

A common source of plan regressions, although by far not the only one, are adjustments of the cost estimation algorithms in an optimizer. Probably the most frequently encountered one is calibration or refinement of the cost model by adjusting certain cost constants used by cost formulas. This type of modification is particularly treacherous because most changes may not affect the BPF of even larger test suites of queries.

To investigate ρ's sensitivity to plan regressions we artificially introduced cost model changes and repeatedly computed the ρ score. We modified the cost function by increasing the cost for hashing an individual datum. Changes to this kind of constant are not unusual in commercial development.

Cost of hash	100%	150%	300%	1000%
ρ	0.919	0.847	0.831	0.557

Table 1: *ρ for Q5 as function of degree of regression introduced in optimizer's cost model; 100% denotes default plan*

In Table 1 we show ρ as a function of the degree of regression introduced for Query Q5. Starting with the standard configuration, i.e., 100% of the cost factor, we increased the cost factor. The correlation coefficient not only reflects the overall trend but helps even quantifying the extent of the regression. Only for the last setting, which leads to significant aberrations in the plan space did the BPF change, i.e., conventional tests would have identified only the last setting as regression.

5.4 Tuning

Our final experiment looks at using ρ to fine-tune individual costing elements. We chose to experiment with two parameters that affect the costing of index seeks: CPU and I/O cost per seek. In Figure 5, ρ scores for combinations of slightly altered cost values are shown. We found that decreasing CPU cost by 10% raises ρ slightly, and decreasing both CPU and I/O cost by 10% to be balanced has significant beneficial value (second and third bar in graph). However, unbalancing these values dramatically by 50% is

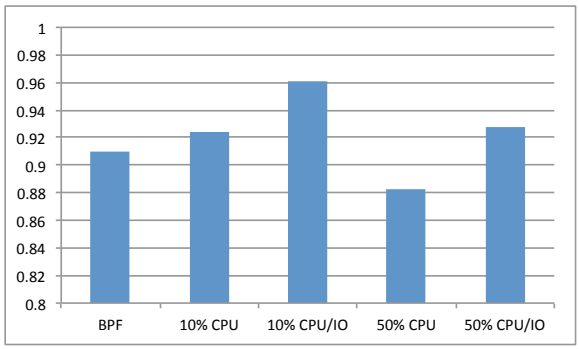


Figure 5: *ρ score for modified index seek costs*

detrimental and may lead to regressions. Again, in neither case was the BPF affected, i.e., conventional testing could not identify the impact of these modifications.

These are only initial experiments that are meant to underline the potential of our methodology but are no comprehensive framework for tuning an optimizer yet. However, we are planning a test harness that combines ρ with automated parameter modifications in the future.

6. DISCUSSION

The initial experiments we presented are a starting point for a variety of additional application areas and research directions. We believe plan space analysis has practical relevance for everybody involved in using, maintaining, or developing query optimization technology. In particular, it can be used by the following groups:

Quality Assurance. QA has been the primary target audience that inspired our work. Testing optimizers poses significant challenges [4]. With a ρ score, regression tests can be easily automated: any randomly generated workload can be turned into a fully functional test suite. Problematic queries can be pin-pointed by their low(er) score. As a result the overall rating of a representative workload can be used to track the quality of the optimizer on a day-by-day basis. Regressions but also improvements are discovered quickly and, most importantly, *early* on in the development process.

Optimizer Implementers. For every software change to the optimizer there exists an infinite set of queries and database configurations for which the search space has been perturbed, i.e., the relative ordering of plans for these queries has been altered. If the choice of the optimal plan of any of these queries has been negatively affected the change is said to have caused a plan regression. The fear of plan regressions is commonly viewed as the biggest obstacle to innovation in optimizer technology. Using a ρ score implementers can make changes more confidently: if the score has degraded the change must have caused one or more plan regressions; if the score has improved the change fixed existing plan regressions, even if the change was not developed as fix for this particular plan regression.

Operations. When upgrading to a new product release plan regressions pose a significant risk that is hard to assess; using the ρ score of the previous and the new release on the customer's own workload allows staff to upgrade with confidence.

DBA's/Data Architects. During the implementation of data warehouses or other application scenarios, data architects or DBA's can test the robustness of individual queries, i.e., assess how well the optimizer can deal with a given query and hence assess the risk for follow up support costs.

7. SUMMARY

Accurately assessing the quality and robustness of a query optimizer is of immense practical relevance and has immediate application in both industry and academic research.

In particular, in this paper we focused on the problem of assessing the impact of changes to the optimizer logic with much finer sensitivity than conventional tests have been able to.

We developed a methodology for a comprehensive plan space analysis. We measure how well a given optimizer models a query's plan space. It is a simple and intuitive measure that assesses the optimizer's ability to rank plan alternatives correctly. Its simplicity, portability, and accuracy make it a universal tool for practitioners in every phase of the development of a query optimizer.

We presented preliminary results of experiments that underline the usefulness of plan space analysis in general and its sensitivity to regressions in the optimizer in particular.

Acknowledgements

We would like to thank the participants of the Dagstuhl workshop on Robust Query Processing for numerous discussions and their encouragement to write this paper.

8. REFERENCES

[1] R. Ahmed. Query Processing in Oracle DBMS. In *Proc. Int'l. Workshop on Data Warehousing and OLAP*, 2010.

[2] M. Elhemali and L. Giakoumakis. Unit Testing Query Transformation Rules. In *Proc. Int'l. Workshop on Database Testing (DBTest)*, 2008.

[3] H. G. Elmongui, V. Narasayya, and R. Ramamurthy. A Framework for Testing Query Transformation Rules. In *Proc. Int'l. Conf. ACM SIGMOD*, 2009.

[4] L. Giakoumakis and C. Galindo-Legaria. Testing SQL Servers Query Optimizer: Challenges, Techniques and Experiences. *IEEE Data Engineering Bulletin*, 31(1), 2008.

[5] G. Graefe, A.-C. König, H. Kuno, V. Markl, and K.-U. Sattler. Robust Query Processing. Technical Report Workshop 10381, Dagstuhl, 2010.

[6] J. S. Maritz. *Distribution-Free Statistical Methods.* Chapman & Hall, 1981.

[7] N. Reddy and J. Haritsa. Analyzing Plan Diagrams of Database Query Optimizers. In *Proc. Int'l. Conf. on Very Large Databases*, 2005.

[8] F. Waas and C. A. Galindo-Legaria. Counting, Enumerating, and Sampling of Execution Plans in a Cost-Based Query Optimizer. In *Proc. Int'l. Conf. ACM SIGMOD*, 2000.