PLAN DIAGRAMS: Visualizing Database Query Optimizers

Jayant R. Haritsa[1]

Abstract The automated optimization of declarative SQL queries is a classical problem that has been diligently addressed by the database community over the last few decades. However, due to its inherent complexities and challenges, this area has largely remained a "black art", and the quality of the query optimizer continues to be a key differentiator between competing database products, with large research and development teams involved in their design and implementation.

Over the past five years, we have provided a fresh perspective on the behavior of modern query optimizers through the introduction and development of the "plan diagram" concept. A plan diagram is a *visual* representation of the plan choices made by the optimizer over a parameter space. In this paper, we provide an overview of plan diagrams, their processing, and their applications.

Plan diagrams often appear similar to cubist paintings and provide a variety of interesting insights, including that current optimizers make extremely fine-grained plan choices; that the plan optimality regions may have highly intricate patterns and irregular boundaries, indicating strongly non-linear cost models; that non-monotonic cost behavior exists where increasing result cardinalities decrease the estimated cost; and, that the basic assumptions underlying parametric query optimization often do not hold in practice.

Our study also shows that complex plan diagrams can usually be reduced to "anorexic" equivalents, featuring only a few plans, without materially affecting the query processing quality. Anorexic reduction has important implications for the design and use of next-generation query optimizers, and in particular, can be used to minimize the adverse impact of selectivity estimation errors, a chronic problem in the field.

Keywords: query optimization, plan diagrams, robust query processing

1 Introduction

Organizations typically collect vast quantities of data relating to their operations. For example, the Income Tax department in India has accumulated a huge repository of information pertaining to taxpayer returns. In order to provide a convenient and efficient environment to productively use these enormous data collections, software packages called "Data Base Management Systems" **(DBMS)** have been developed and refined over the past half-century, beginning in the early 1960s. These packages are now extensively used throughout the world in virtually the entire spectrum of human activity including banking, insurance, governance, business, travel, manufacturing, education and health-care. Popular commercial offerings of DBMS software include IBM's DB2 [DB2], Oracle [ORACLE], Microsoft's SQL Server [SQLSERVER] and Sybase ASE [SYBASE], while PostgreSQL [POSTGRES] and MySQL [MYSQL] are well-known public-domain packages.

A primary reason for the popularity of DBMS is that they provide a simple but powerful logical model wherein all data is stored in the form of tables called *relations*, supported by friendly and powerful interfaces to ask questions, called *queries*, on the information stored in these database relations. Queries are usually expressed in the Structured Query Language (SQL) [SQL], the global de facto standard for interfacing with relational database systems. A particularly appealing feature of the language is that it is "declarative", meaning that the user only states *what* is wanted without having to specify the *procedure* for obtaining the information.

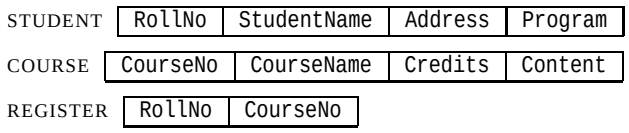


Figure 1: University Database

```
select StudentName, CourseName
from STUDENT, COURSE, REGISTER
where STUDENT.RollNo = REGISTER.RollNo and
   REGISTER.CourseNo = COURSE.CourseNo
```

Figure 2: Example SQL Query

To make the declarative notion concrete, consider the sample university database shown in Figure 1. Here, information is maintained in three relations: STUDENT, COURSE and REGISTER, which tabulate data about students, courses, and the course registrations of students, respectively. The user's goal is to extract the names of the students and the courses for which they are registered, and a sample SQL query that achieves this goal is shown in Figure 2, where the desired information is obtained by combining the data across the three tables using the roll numbers and the course numbers as the connectors. Note that in this formulation, the *sequence* in which the tables are combined, as well as the *mechanism* to be used for each combination is left unspecified, resulting in the declarative tag.

[1]Supercomputer Education & Research Centre and Dept. of Computer Science & Automation, Indian Institute of Science, Bangalore 560012.

1.1 Query Optimization

Given a generic SQL query that requires combining information across *n* relations, there are in principle *n!* different permutations of the combination sequence, implying that the strategy search space is at least *exponential* in the query size. The automated identification of an efficient procedure or strategy from this search space is the responsibility of an internal DBMS component called the **"query optimizer"**. The efficiency of these strategies, called "plans", is usually costed in terms of the estimated query response time. Optimization is a mandatory exercise since the difference between the cost of the best plan and a random choice could be in orders of magnitude, but is computationally extremely expensive due to the combinatorially large search space of plan alternatives, as explained above. The role of query optimizers has become especially critical in recent times due to the high degree of query complexity characterizing current decision-support applications, as exemplified by the industry-standard TPC-H and TPC-DS performance benchmarks [TPCH; TPCDS].

Plans are typically comprised of a *tree* of data processing operators that are evaluated in a bottom-up paradigm. A sample plan is shown in Figure 3 for the example query of Figure 2, where the STUDENT and REGISTER relations are first combined with a NESTED-LOOP join operator, and this intermediate result is then combined with the COURSE relation using a HASH-JOIN operator. The bracketed numbers within each operator node indicate the estimated aggregate processing costs incurred until this stage in the bottom-up query evaluation.

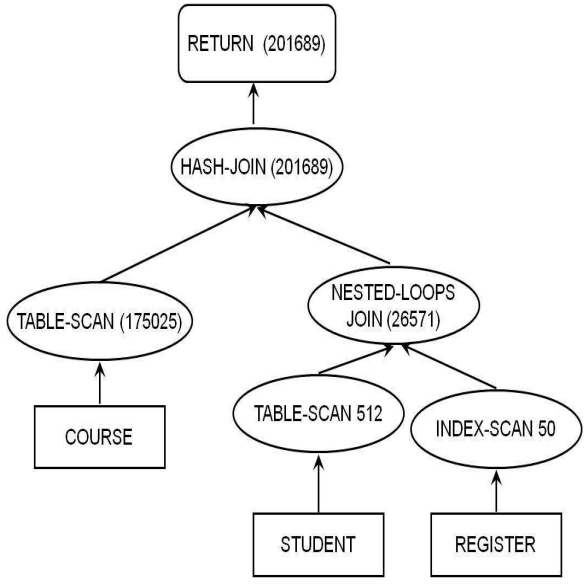


Figure 3: Sample Plan

The design of effective query optimizers that quickly identify low cost plans has been diligently addressed by the database research community over the last few decades [Chaudhuri (1998)]. However, due to its inherent complexities and challenges, this area has largely remained a "black art", and the quality of the query optimizer continues to be a key differentiator between competing database products, with large R & D teams involved in their design and implementation.

Over the past five years, we have been able to provide a fresh perspective on the behavior of modern query optimizers through the introduction and development of the **"plan diagram"** concept. A plan diagram is a *visual* representation of the plan choices made by the optimizer over a parameter space, generated by leveraging the optimizer's Application Programming Interface (API) functions. In the remainder of this paper, we provide an overview of plan diagrams, their processing, and their applications. The complete technical details of the material covered here are available in a suite of papers [Reddy and Haritsa (2005); Harish, Darera and Haritsa (2007); Harish, Darera and Haritsa (2008); Dey, Bhaumik, Harish and Haritsa (2008); Abhirama, Bhaumik, Dey, Shrimal and Haritsa (2010); Haritsa (2010)], published in the Very Large Data Base (VLDB) conference series, a premier international forum for the dissemination of database research.

2 Plan Diagrams

Modern query optimizers each have their own "secret sauce" to identify the best (i.e. cheapest) plan for answering declarative SQL queries. However, the de-facto standard underlying strategy, pioneered by the System R project at IBM Research [Selinger, Astrahan, Chamberlin, Lorie and Price (1979)] is the following: *Given a user SQL query, apply a variety of heuristics to restrict the exponential plan search space to a manageable size; estimate, with a cost model and a dynamic-programming-based processing algorithm, the efficiency of each of these candidate plans; finally, choose the plan with the lowest estimated cost.*

A query optimizer's execution plan choices, for a given database and system configuration, are primarily a function of the *selectivities* of the base relations in the query. The selectivity of a relation is the estimated fraction of rows of the relation that are relevant to producing the final result. In [Reddy and Haritsa (2005)], we introduced plan diagrams to denote color-coded pictorial enumerations of the plan choices of the optimizer for parametrized SQL query templates over the relational selectivity space. For example, consider QT8, the parametrized two-dimensional query template shown in Figure 4, based on Query 8 of the TPC-H benchmark (the query determines the market share of Brazil within the American continent for cheap anodized steel parts). The template has selectivity variations on the SUPPLIER and LINEITEM relations through the s_acctbal :varies and l_ extendedprice :varies predicates, which apply one-sided range constraints on the supplier's account balance and the extended price of the lineitem, respectively.

```
select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from
    (select YEAR(o_orderdate) as o_year, l_extendedprice * (1 - l_discount) as volume,
            n2.n_name as nation
    from part, supplier, lineitem, orders, customer, nation n1, nation n2, region
    where p_partkey = l_partkey and s_suppkey = l_suppkey and l_orderkey = o_orderkey and
       o_custkey = c_custkey and c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey
       and s_nationkey = n2.n_nationkey and r_name = 'AMERICA' and p_type = 'ECONOMY ANODIZED
       STEEL' and s_acctbal :varies and l_extendedprice :varies
    ) as all_nations
group by o_year
order by o_year
```

Figure 4: Example Query Template (QT8)

The associated plan diagram for QT8 is shown in Figure 5(a) – this picture was produced on a commercial database engine using the Picasso visualization software tool developed in our lab [PICASSO]. In this picture, a set of 89 different optimal plans, P1 through P89, cover the selectivity space. The value associated with each plan in the legend indicates the percentage area covered by that plan in the diagram – P1, for example, covers about 22% of the space, whereas P89 is chosen in only 0.001% of the space. In a nutshell, plan diagrams visually capture the geometries of the optimality regions of the parametric optimal set of plans (POSP) [Hulgeri and Sudarshan (2002)].

It is vividly evident from Figure 5(a) that plan diagrams can be extremely complex and dense, with a large number of plans covering the space – several such instances spanning a representative set of query templates over a suite of industrial-strength optimizers, are available at [PICASSO]. In fact, they often appear similar to *cubist paintings*, and hence the name Picasso for our visualization tool.[2] We have identified a variety of intricate tessellated patterns, including *speckles*, *stripes*, *blinds*, *mosaics* and *bands*, in the diagrams. Further, the boundaries of the plan optimality regions can be highly irregular, which seem to indicate the presence of strongly non-linear and discretized cost models. Finally, the diagrams also demonstrate that the basic assumptions – *plan convexity*, *uniqueness* and *homogeneity* – underlying the rich body of research literature on parametric query optimization (e.g. [Hulgeri and Sudarshan (2002); Hulgeri and Sudarshan (2003)]), rarely hold in practice.

While *individual* queries have been analyzed in great detail in the past, our work is among the first to characterize and investigate the behavior over a *parameter space* in an industrial-strength environment. Therefore, in spite of query optimization having been studied for several decades, our discovery of the above-mentioned complex patterns has proved to be rather surprising and thought-provoking for the database research community.

Plan diagrams are currently in vogue at various industrial and academic sites for a diverse set of applications including analysis of existing optimizer designs; visually carrying out optimizer regression testing; debugging new query processing features; comparing the behavior between different optimizer versions; investigating the structural differences between neighboring plans in the space; evaluating the variations in the plan choices made by competing optimizers; etc. As a case in point, visual examples of *non-monotonic* cost behavior in commercial optimizers, potentially indicative of modeling errors, were highlighted in [Reddy and Haritsa (2005)].

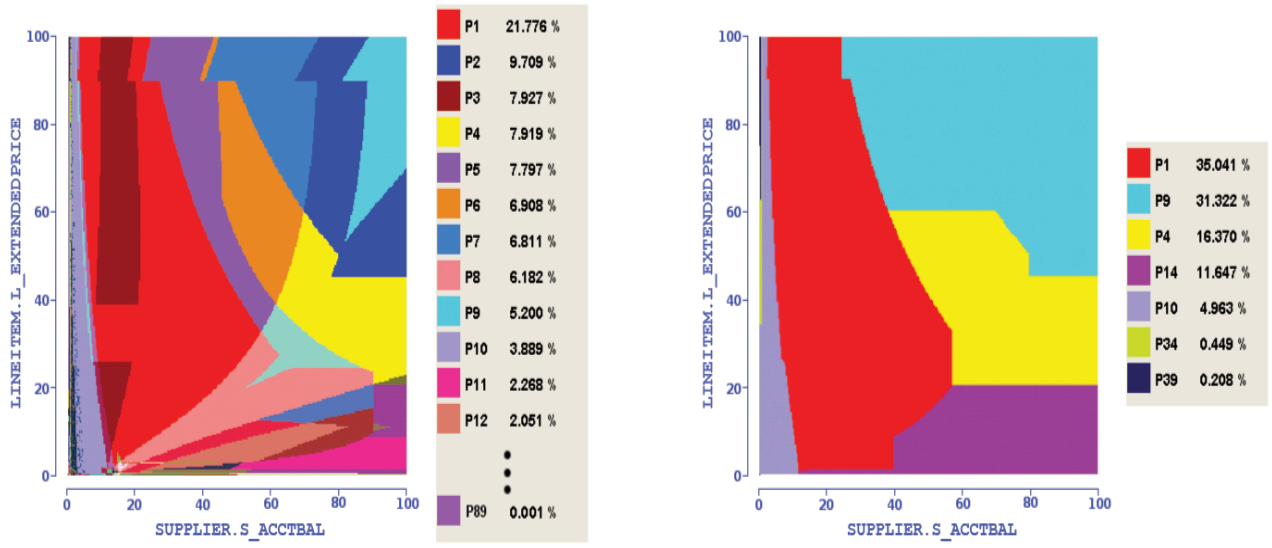
[2] Pablo Picasso is considered to be a founding-father of the cubist genre of painting [CUBIST].

2.1 Anorexic Reduction of Plan Diagrams

In the next phase of our investigation, we showed that dense plan diagrams could typically be "reduced" to much simpler pictures featuring significantly fewer plans, *without materially degrading the processing quality of any individual query*. For example in Figure 5(a), if users are willing to tolerate a minor cost increase, denoted by λ, of at most 10% for any query in the diagram, relative to its original cost, the picture could be reduced to Figure 5(b), where only 7 plans remain – that is, most of the original plans have been "completely swallowed" by their siblings, leading to a highly reduced plan cardinality.

We presented in [Harish, Darera and Haritsa (2007)] a detailed study of the plan diagram reduction problem from theoretical, statistical and empirical perspectives. Our analysis first showed that finding the optimal (wrt minimizing the number of plans) reduced plan diagram is NP-Hard through a reduction from the classical Set Cover problem [Garey and Johnson (1979)]. This result motivated the design of **CostGreedy**, a greedy heuristic algorithm whose complexity is $O(nm)$, where n is the number of plans and m is the number of query points in the diagram ($n \ll m$). Hence, for a given picture resolution, CostGreedy's performance scales *linearly* with the number of plans in the diagram. Further, from the reduction quality perspective, CostGreedy provides a tight performance guarantee of $O(\ln m)$, which cannot be improved upon by any other deterministic algorithm.

We also considered a storage-constrained variant of the plan



(a) **Plan Diagram** (b) **Reduced Diagram (λ = 10%)**

Figure 5: Sample Plan Diagram and Reduced Plan Diagram (QT8)

diagram reduction problem, wherein at most k plans can be retained for a query template, and the objective is to find the k plans that minimize the maximum cost increase of the query points in the reduced plan diagram. We showed that this variant retains the hardness of the general problem. On the positive side, however, we were able to design **ThresholdGreedy**, a greedy algorithm that delivers a performance guarantee of 0.63 relative to the optimal.

Then, using extremely coarse characterizations of the cost distributions of the optimal plans, we developed fast but effective estimators for determining the expected number of plans retained for a given threshold. These estimators can also be used to predict the location of the best possible trade-off (i.e. the "knee") between the plan cardinality reduction and the cost-increase threshold.

Lastly, through experimental analysis on the plan diagrams produced by industrial-strength optimizers with benchmark-based multi-dimensional query templates, we showed that: (a) plan reduction can be carried out efficiently, since our attention is limited to only the set of plans appearing in the original plan diagram, making it unnecessary to revisit the optimizer's combinatorially large search space of plan alternatives; (b) the CostGreedy algorithm typically gives the optimal reduction or is within a few plans of the optimal; and (c) the analytical estimates of the plan-reduction versus cost-threshold graph are quite accurate.

Most importantly, our results demonstrated that a cost-increase threshold of *only 20 percent* is usually amply sufficient to bring down the absolute number of plans in the final reduced picture to *within or around ten*. In short, that complex plan diagrams can be made "anorexic" while retaining acceptable query processing performance, even for high dimensional query templates.

Carrying out anorexic plan reduction on dense plan diagrams has a variety of useful implications for improving both the efficiency of the optimizer and the choice of execution plan. These applications include, as described in detail in [Harish, Darera and Haritsa (2007)], quantifying the redundancy in the plan search space, enhancing the usability of parametric query optimization techniques [Hulgeri and Sudarshan (2002), Hulgeri and Sudarshan (2003)], and identifying error-resistant and least-expected-cost plans [Chu, Halpern and Seshadri (1999); Chu, Halpern and Gehrke (2002); Babcock and Chaudhuri (2005)].

Anorexic reduction's most important utility, however, as described in the following subsection, is that it supports the identification of plans that are *robust* to errors in selectivity estimates, a chronic problem faced by database query optimizers, arising due to a variety of reasons, including outdated statistics, attribute-value independence assumptions, and coarse summaries [Stillger, Lohman, Markl and Kandil (2001)]. To address this problem, an obvious approach is to improve the quality of the statistical meta-data, for which several techniques have been presented in the literature ranging from improved summary structures [Chaudhuri and Aboulnaga (1999)] to feedback-based adjustments [Stillger, Lohman, Markl and Kandil (2001)] to on-the-fly re-optimization of queries [Kabra and DeWitt (1998); Markl, Raman, Simmen, Lohman, Pirahesh and Cilimdzic (2004); Babu, Bizarro and DeWitt (2005)]. However, the complementary

and conceptually different approach considered in our work is to identify *robust plans* that are relatively less sensitive to such selectivity errors. In a nutshell, to "aim for resistance, rather than cure", by identifying plans that provide comparatively good performance over large regions of the selectivity space. Such plan choices are especially important for industrial workloads where global stability is as much a concern as local optimality [Mackert and Lohman (1986)] .

2.2 Selecting Robust Plans

In [Harish, Darera and Haritsa (2008)], we addressed the selectivity-error issue from both theoretical and empirical perspectives. Through extensive experimentation on a leading commercial optimizer with a rich suite of multi-dimensional benchmark-based query templates operating on a variety of logical and physical database designs, we demonstrated that *plan diagram reduction typically produces plan choices that substantially curtail the adverse effects of selectivity estimation errors*. Therefore, it clearly has potential to improve performance in general, especially for errors that lie within the swallower's optimality region, i.e. its "endo-optimal" region.

Consider a query instance whose optimizer-estimated location in the selectivity space is q_e, and denote the optimizer's optimal plan choice at q_e by P_{oe}. Due to errors in the selectivity estimates, the *actual* location of q_e could be different at execution-time – denote this location by q_a, and the optimizer's optimal plan choice at q_a by P_{oa}. Assume that P_{oe} has been swallowed by a sibling plan during the reduction process and denote the replacement plan assigned to q_e by P_{re}.

Replacement Benefits. Our first scenario, typical of that seen in most of our experiments, demonstrates how the replacement plan P_{re} can provide extremely substantial improvements *throughout the selectivity space*. Specifically, on vanilla hardware with a popular commercial optimizer, we generated a plan diagram for a 2D query template based on TPC-H Q5, with selectivity variations on the CUSTOMER and SUPPLIER relations, and carried out reduction with $\lambda = 10\%$. On this diagram, consider the estimated location $q_e = (0.36, 0.05)$ and a sample set of actual locations q_a – for instance, along the principal diagonal of S. For this scenario, the costs of P_{oe} (P45), P_{re} (P17) and P_{oa} (the optimal plan at each q_a location) are shown in Figure 6 – note that the costs are measured on a *log scale*.

It is clear from Figure 6 that the replacement plan P_{re} provides *orders-of-magnitude* benefit with respect to P_{oe}. In fact, the error-resistance is to the extent that it virtually provides "immunity" to the error since the performance of P_{re} is close to that of the *locally optimal plan* P_{oa} throughout the space – note that is in spite of the endo-optimal region of P_{re} constituting only a very small fraction of this space.

To demonstrate that the benefits anticipated from the compile-time analysis do translate to corresponding improvements at *run-time*, we also explicitly evaluated the query response times at the same q_a locations. We found that huge savings in processing time were obtained by using the replacement plan instead of the optimizer's original choice, and that the replacement's performance is virtually indistinguishable from the optimal choices.

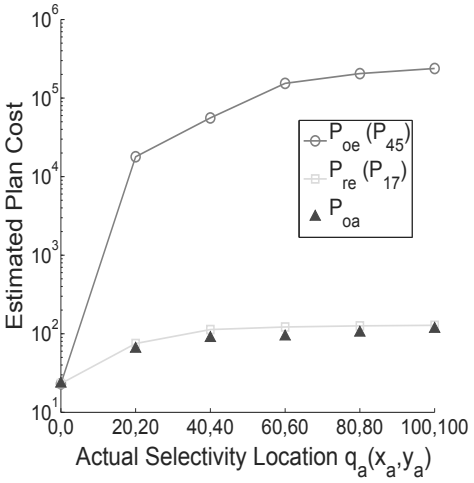


Figure 6: Beneficial Impact of Plan Replacement

Replacement Problems. While performance improvements are usually the order of the day, we also encountered occasional situations wherein a replacement plan performs much worse in its exo-optimal region than the original optimizer choice, that is, where P_{re} performs worse than P_{oe} at q_a. A particularly egregious example, arising from the *same* plan diagram described above, is shown in Figure 7 for $q_e = (0.03, 0.14)$ – we see here that it is now the replacement plan P_{re} (P34), which is *orders-of-magnitude* worse than P_{oe} (P26) in the presence of selectivity errors. This compile-time assessment was also corroborated by evaluating the corresponding query response times.

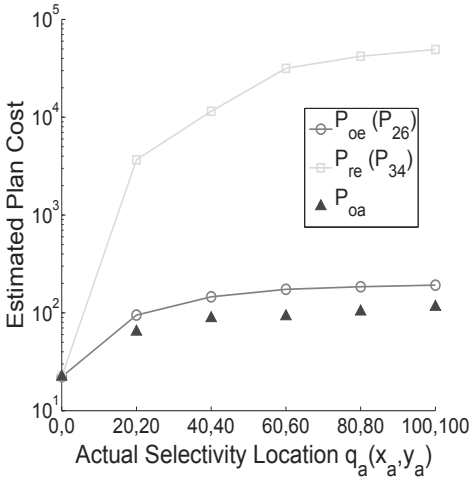


Figure 7: Adverse Impact of Plan Replacement

The above example highlights the need to establish an efficient criterion of when a specific swallowing is *globally safe*, that is, within the λ-threshold throughout the space. To achieve this objective, we designed a generalized mathematical model of the behavior of plan cost functions over the selectivity space. The model, although simple, is sufficient to capture the cost behavior of all plans that have arisen from our query templates, and is the first such characterization for industrial-strength optimizers. Specifically, the cost model of a plan in a 2D selectivity space is of the form

$$cost(x,y) = a_1 x + a_2 y + a_3 xy + a_4 x\log x + a_5 y\log y + a_6 xy\log xy + a_7$$

where $a_1, a_2, a_3, a_4, a_5, a_6, a_7$ are coefficients, and x, y represent the selectivity dimensions, respectively. Modeling a specific plan requires suitably choosing the seven coefficients in the above equation.

Using this model, we then proved the powerful result that safety checks on only the *perimeter* of the selectivity space are sufficient to decide the safety of reduction over the *entire* space. These checks involve the costing of *"foreign plans"*, that is, of costing plans in their exo-optimal regions, a feature that has become available in the current versions of several industrial-strength optimizers. Apart from providing reduction safety, foreign-plan costing was additionally leveraged to both (a) enhance the degree of reduction of the plan diagram, and (b) improve the complexity characteristics of the reduction process, as compared to the earlier CostGreedy algorithm [Harish, Darera and Haritsa (2007)].

Overall, our new approach called **SEER** (Selectivity-Estimate-Error-Resistance), provides an effective and safe mechanism for identifying robust plans that are resistant, as compared to the optimizer's original choices, to errors in the base relation selectivity estimates. We have also developed LiteSEER, an optimally-efficient light-weight heuristic version of SEER that very cheaply provides a high degree of safety by restricting its attention to only the *corners* of the selectivity space.

A particularly noteworthy aspect of our techniques is that their performance guarantees apply at the level of *individual queries*. This is in marked contrast to the *aggregate* basis of prior proposals in the literature, which made them difficult to use in practice. Further, since we treat the optimizer as a black-box, our approach is inherently (a) completely non-intrusive, and (b) capable of handling whatever SQL is supported by the system. Equally importantly, we do not expect to have any additional information beyond that provided by the engine's API interface.

Viewed in toto, the results presented in [Harish, Darera and Haritsa (2008)] indicate that a large percentage of optimizer choices over the parameter space can be improved through robust replacements.

2.3 Efficient Generation of Plan Diagrams

Plan diagrams have proved to be a powerful metaphor for the analysis and redesign of modern optimizers, and are gaining currency in diverse industrial and academic institutions. However, their utility is adversely impacted by the impractically large computational overheads incurred when standard brute-force exhaustive approaches are used for producing fine-grained diagrams on high-dimensional query templates.

For example, a 2D plan diagram with a resolution of 1000 on each selectivity dimension, or a 3D plan diagram with a resolution of 100 on each dimension, both require invoking the optimizer a *million* times. Even with a conservative estimate of about a half-second per optimization, the total time required to produce the picture is close to a week! Therefore, although plan diagrams have turned out to be extremely useful, their high-dimension and/or fine-resolution versions pose serious computational challenges.

In [Dey, Bhaumik, Harish and Haritsa (2008)], we investigated strategies for efficiently producing close *approximations* to complex plan diagrams. Our techniques are customized to the features available in the optimizer's API, ranging from the generic optimizers that provide only the optimal plan for a query, to those that also support costing sub-optimal plans, and enumerating rank-ordered lists of plans. The techniques collectively feature both random and grid sampling, as well as interpolation techniques based on nearest-neighbor classifiers, parametric query optimization and plan cost monotonicity.

Extensive experimentation with benchmark-based query templates on industrial-strength optimizers indicates that our techniques are capable of delivering 90% accurate diagrams while incurring less than 15% of the computational overheads of the exhaustive approach. In fact, for full-featured optimizers that support the production of the second-best plan in addition to the cheapest plan, we have been able to devise a strategy called **DiffGen** that can be be formally proved to guarantee *zero error*, while incurring only around 10% overheads. A variant of DiffGen, called **ApproxDiffGen**, trades error for further reduction in optimization effort and produces 90% accurate diagrams with less than 5% overheads.

2.4 Run-time Applications

Apart from aiding optimizer design, plan diagrams can also be used in *operational* settings. Specifically, since they identify the optimal set of compile-time plans, they can be used at run-time to immediately identify the best plan for the current query without going through the time-consuming optimization exercise. Further, they can prove useful to *adaptive* plan selection techniques (e.g. [Antoshenkov (1993); Babu, Bizarro and DeWitt (2005); Cole and Graefe (1994); Deshpande, Ives and Raman (2007); Kabra and DeWitt (1998); Markl, Raman, Simmen, Lohman, Pirahesh and Cilimdzic (2004)]), which, based on run-time observations, may dynamically choose to

re-optimize the query and switch plans mid-way through the processing. In this context, plan diagrams can help to eliminate the re-optimization overheads incurred in determining the substitute plan choices. The reduced plan diagrams, on the other hand, help to both minimize the number of invocations of the re-optimization process, as well as the likelihood of requiring a plan switch after the re-optimization.

3 Online Robust Plans

The SEER algorithm for identifying robust plans, described in the previous section, is an *off-line* approach in that it uses prior knowledge of the POSP set of plans in order to make the replacements. The scheme operates from outside the optimizer, treating it as a black box that supplies plan-related information through its API, and is usable only on form-based query templates for which plan diagrams have been previously computed.

In practice, however, we would ideally like to have the robustness feature to be *organically* integrated within the optimizer, rather than generated as a post-facto exercise, and by virtue of this integration, making it directly applicable to ad-hoc individual queries. We have achieved this goal in our most recent work [Abhirama, Bhaumik, Dey, Shrimal and Haritsa (2010)] through an algorithm called **EXPAND**, which judiciously expands the candidate set of sub-plans that are retained at each node of the plan enumeration lattice during the core dynamic-programming exercise. That is, instead of merely forwarding the cheapest sub-plan from each node in the lattice, a *train* of sub-plans is sent, with the cheapest being the "engine", and stabler alternative choices being the "wagons". To ensure that the overheads of maintaining trains instead of engines are not impractically large, a four-stage pruning process that incorporates both cost and robustness aspects is used to ensure that only wagons that are plausible replacements for the engine are retained. The final plan selection is made at the root of the dynamic-programming lattice from amongst the set of complete plans available at this terminal node, subject to user-specified cost and stability criteria.

The Expand scheme has been incorporated in the kernel of the public-domain PostgreSQL database engine, and a variety of plan selection algorithms that cover a spectrum of design tradeoffs have been been implemented and evaluated on benchmark environments. Our results have shown that a significant degree of robustness can be obtained with relatively minor conceptual changes to current optimizers, especially those supporting a foreign-plan-costing feature. More specifically, while incurring additional time overheads within 100 milliseconds, and memory overheads within 100MB, Expand often delivers plan choices that eliminate more than *two-thirds of the performance gap* (between P_{oe} and P_{oa}) for a non-trivial number of error instances. Equally importantly, the replacement is almost never materially worse than the optimizer's original choice. In a nutshell, our replacement plans *"often help substantially, but never seriously hurt"* the query performance.

Overall, the Expand approach results in an intrinsically improved optimizer that directly and efficiently produces high-quality plan diagrams in an online fashion, rather than as a post-processing step.

4 Closing Remarks

Database engines pervade all aspects of our lives, and therefore, their design and analysis forms a core component of the computer science discipline. In our lab at IISc, we have addressed fundamental research challenges in the development of these engines for a variety of application contexts. Our work over the past five years on industrial-strength database query optimization has garnered attention from both the academic and industrial communities, and the Picasso tool, which provides a powerful visual metaphor fuelled by high-performance algorithms, is fast becoming an integral part of the design workbench in database companies and university labs worldwide.

From the perspective of system developers and practitioners, the concepts and diagrams presented in our work can serve as potent mechanisms for the analysis, testing and redesign of their systems. In fact, plan diagrams have been prominently featured in recent papers by leading industry experts in query optimization [Graefe, Kuno and Wiener (2009); Chaudhuri (2009)]. Further, the dire need and importance of research on robust query processing is testified to by the just-concluded Dagstuhl Seminar 10381 [DAGSTUHL], which was devoted solely to this topic, and where it was mentioned that *lack of robustness can contribute as much as a third to the total cost of ownership for a database system*.

Finally, for database instructors and students, plan diagrams are an intuitive pedagogical support to help comprehend and appreciate the complexities and subtleties of industrial-strength query optimization. The diagrams represent a significant departure from the toy textbook examples that are typically covered in a classroom setting.

Overall, the primary message of our work is that it is indeed feasible to *efficiently produce plan diagrams that simultaneously possess the highly desirable properties of being online, anorexic, safe and robust*. We expect that this result will have a significant impact on the design of next-generation database query optimizers.

Acknowledgements. This work was supported in part by a Swarnajayanti Fellowship from the Dept. of Science & Technology, Govt. of India, a research grant from the Dept. of Bio-technology, Govt. of India, and research grants from a host of multinational IT companies, including IBM, Google and Microsoft.

References:

Abhirama, M; Bhaumik, S; Dey, A; Shrimal, H; Haritsa, J. (2010) On the Stability of Plan Costs and the Costs of Plan Stability, *PVLDB Journal*, 3(1).

Antoshenkov, G. (1993) Dynamic Query Optimization in Rdb/VMS, *Proc. of 9th IEEE Intl. Conf. on Data Engineering (ICDE)*, pp 538-547.

Babcock, B; Chaudhuri, S. (2005) Towards a Robust Query Optimizer: A Principled and Practical Approach, *Proc. of ACM SIGMOD Conf. on Management of Data*, pp 119-130.

Babu, S; Bizarro, P; DeWitt, D. (2005) Proactive Re-Optimization, *Proc. of ACM SIGMOD Conf. on Management of Data*, pp 107-118.

Chaudhuri, S. (1998) An Overview of Query Optimization in Relational Systems, *ACM Symp. on Principles of Database Systems (PODS)*, pp 34-43.

Chaudhuri, S. (2009) Query Optimizers: Time to Rethink the Contract?, *Proc. of ACM SIGMOD Conf. on Management of Data*, pp 961-968.

Chaudhuri, S; Aboulnaga, A. (1999) Self-tuning Histograms: Building Histograms without Looking at Data, *Proc. of ACM SIGMOD Conf. on Management of Data*, pp 181-192.

Chu, F; Halpern, J; Gehrke, J. (2002) Least Expected Cost Query Optimization: What Can We Expect, *Proc. of ACM Symp. on Principles of Database Systems (PODS)*, pp 293-302.

Chu, F; Halpern J; Seshadri, P. (1999) Least Expected Cost Query Optimization: An Exercise in Utility, *Proc. of ACM Symp. on Principles of Database Systems (PODS)*, pp 138-147.

Cole, R; Graefe, G. (1994) Optimization of Dynamic Query Evaluation Plans, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, pp 150-160.

CUBIST `www.artlex.com.h/ArtLex/c/cubism.html`

DAGSTUHL `www.dagstuhl.de/no_cache/en/program/calendar/semhp/?semnr=10381`

DB2 `www.ibm.com/db2`

Deshpande, A; Ives, Z; Raman, V. (2007) Adaptive Query Processing, *Foundations and Trends in Databases*, Now Publishers, 1 (1).

Dey, A; Bhaumik, S; Harish, D; Haritsa, J. (2008) Efficiently Approximating Query Optimizer Plan Diagrams, *PVLDB Journal*, 1(2), pp. 1325-1336.

Garey, M; Johnson, D (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W H Freeman & Co.

Graefe, G; Kuno, H; Wiener, J. (2009) Visualizing the robustness of query execution, *Proc. of 4th Conf. on Innovative Data Systems Research (CIDR)*.

Harish, D; Darera, P; Haritsa, J. (2007) On the Production of Anorexic Plan Diagrams, *Proc. of 33th Intl. Conf. on Very Large Data Bases (VLDB)*, pp 1081-1093.

Harish D;, Darera, P; Haritsa, J. (2008) Identifying Robust Plans through Plan Diagram Reduction, *PVLDB Journal*, 1(1), pp 1124-1140.

Haritsa, J. (2010) The Picasso Database Query Optimizer Visualizer, *PVLDB Journal*, 3(2).

Hulgeri, A; Sudarshan, S. (2002) Parametric Query Optimization for Linear and Piecewise Linear Cost Functions, *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB)*, pp 167-178.

Hulgeri, A; Sudarshan, S. (2003) AniPQO: Almost Non-intrusive Parametric Query Optimization for Nonlinear Cost Functions, *Proc. of 29th Intl. Conf. on Very Large Data Bases (VLDB)*, pp 766-777.

Kabra, N; DeWitt, D. (1998) Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, pp 106-117.

Mackert, L; Lohman, G. (1986) R^* Optimizer Validation and Performance Evaluation for Local Queries, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, pp 84-95.

Markl, V; Raman, V; Simmen, D; Lohman, G; Pirahesh, H; Cilimdzic, M. (2004) Robust Query Processing through Progressive Optimization, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, pp 659-670.

MYSQL `www.mysql.com`

ORACLE `www.oracle.com/technology/products/database/oracle11g/`

PICASSO `dsl.serc.iisc.ernet.in/projects/PICASSO/picasso.html`

POSTGRES `www.postgresql.org`

Reddy, N; Haritsa J. (2005) Analyzing Plan Diagrams of Database Query Optimizers, *Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB)*, pp 1228-1240.

Selinger, P; Astrahan, M; Chamberlin, D; Lorie, R; Price, T. (1979) Access Path Selection in a Relational Database System, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, pp 23-34.

SQL `en.wikipedia.org/wiki/SQL:2008`

SQLSERVER `www.microsoft.com/sqlserver/2008/`

Stillger, M; Lohman, G; Markl, V; Kandil, M. (2001) LEO – DB2's LEarning Optimizer, *Proc. of 27th Intl. Conf. on Very Large Data Bases (VLDB)*, pp 19-28.

SYBASE `www.sybase.com/linux/ase`

TPCDS `www.tpc.org/tpcds`

TPCH `www.tpc.org/tpch`