Foundations and Trends® in Databases

Robust Query Processing: A Survey

Suggested Citation: Jayant R. Haritsa (2024), "Robust Query Processing: A Survey", Foundations and Trends® in Databases: Vol. 15, No. 1, pp 1–114. DOI: 10.1561/1900000089.

Jayant R. Haritsa
Indian Institute of Science
haritsa@iisc.ac.in

now
the essence of knowledge
Boston — Delft

To my paternal and maternal grandfathers, Prof. T. S. Subbaraya and Prof. B. N. Balakrishna Rao, beacons of intellectual dedication.

To Seema, Tulasi and Sarangi, infusing life with melody.

Contents

Robust Query Processing: A Survey

Jayant R. Haritsa

Indian Institute of Science (IISc) Bangalore, India; haritsa@iisc.ac.in

ABSTRACT

The primordial function of a database system is to efficiently compute correct answers to user queries. Therefore, robust query processing (RQP), where strong numerical guarantees are provided on query performance, has been a long-standing core objective in the design of industrial-strength database engines. Unfortunately, however, RQP has proved to be a largely intractable and elusive challenge, despite sustained efforts spanning several decades. This problematic situation has arisen from a variety of knotty technical hurdles, including complex query representations, limited metadata coverage, coarse statistical models, and hypersensitive operator behaviors. Its impact is felt acutely since the performance degradation faced by database queries can be huge, reaching orders of magnitude as compared to an oracular ideal.

Notwithstanding this daunting history, the good news is that in recent times, there have been a host of exciting technical advances that collectively promise to materially address the robustness objective. The new approaches have been constructed at different levels in the database architecture, and tackle robustness in cost models, database operators, query execution plans and query processing strategies. Although most of this literature is based on statistical and geometric formulations, a significant corpus of machine learning-based techniques is also now available.

Jayant R. Haritsa (2024), "Robust Query Processing: A Survey", Foundations and Trends® in Databases: Vol. 15, No. 1, pp 1–114. DOI: 10.1561/1900000089.

In this monograph, we present an overview of these novel research paradigms, and highlight their strengths and limitations. Further, we enumerate a suite of open technical problems that remain to be solved to make RQP a contemporary reality.

1

Introduction

An organic reason for the ubiquitous popularity of database management systems is their support for *declarative* user queries, typically expressed in SQL. In this framework, the user only specifies the *end* objectives, leaving it to the database system to first identify and then execute the most efficient *means*, called "plan", to achieve these objectives. The identification and execution steps are performed by the *query optimizer* and *query executor* components, respectively, within the core of the database engine. Over the past half century, research on the design and implementation of these components has been a foundational topic for both the academic and industrial database communities.

A de facto global consensus exists on the technologies underlying most of the core database engine modules – for instance, two-phase locking (2PL) for concurrency control, write-ahead logging (WAL) for database recovery, least recently used (LRU-K) for memory management, and a combination of Bitmaps and B-trees for indexing database columns. Therefore, one might expect that a similar situation holds for query processing as well. However, despite the decades of research mentioned above, the unfortunate reality is that the proposed solutions have largely remained a "black art". This is due to the well-documented

complexities and challenges of database query processing (Chaudhuri, 1998; Chaudhuri, 2009), which include complex query representations, limited metadata coverage, coarse statistical models, and hypersensitive operator behavior.

In fact, the prevalent situation is dire enough that a highly respected industry veteran was provoked to lament (Lohman, 2014): *The wonder isn't "Why did the optimizer pick a bad plan?" Rather, the wonder is "Why would the optimizer ever pick a decent plan?"*! He ended with the following exhortation to the research community: *"Let's attack problems that really matter, those that account for optimizer disasters, and stop polishing the round ball."* Similar sentiments have also been expressed by other academic and industrial database experts, including: *"Query optimizers do a terrible job of producing reliable, good plans (for complex queries) without a lot of hand tuning"* (Winslett, 2002), and *"Almost all of us who have worked on query optimization find the current state of the art unsatisfactory with known big gaps in the technology"* (Parameswaran, 2012).

What makes this parlous state of affairs particularly problematic is that the performance degradation faced by database queries can be *huge* – often in *orders of magnitude*, as compared to an oracular ideal that magically knows the correct inputs required for optimal query processing. As a case in point, when Query 19 of the TPC-DS benchmark (Transaction Processing Council, 2024a) is executed on a popular industrial-strength database system, the worst-case slowdown, relative to the hypothetical oracle, can exceed a *million*! (Dutt and Haritsa, 2016). Moreover, apart from the obvious detrimental impacts on user productivity and satisfaction, there are also adverse financial implications: the total cost of ownership is significantly increased due to over-provisioning, lost efficiency, and increased human administrative costs (Wiener *et al.*, 2009).

In the midst of this gloom and doom, the positive news is that in recent times there have been a host of exciting research advances, which collectively promise to provide strong foundations for designing the next generation of query processing engines. The new approaches have been constructed at different levels in the database architecture, and tackle robustness in cost models, database operators, query execution

plans and query processing strategies. Although most of this literature is based on statistical and geometric formulations, a significant corpus of machine learning-based techniques has also now become visible.

The expectation is that these advances will eventually organically support **robust query processing (RQP)** with strong performance guarantees, relegating to the past the above-mentioned cynicism on this bedrock objective. Many of the new ideas owe their genesis to a series of influential and well-attended Dagstuhl Seminars on the topic of RQP over the past decade (Graefe *et al.*, 2010; Graefe *et al.*, 2012; Borovica-Gajic *et al.*, 2017; Böhm *et al.*, 2021). Further, they have arisen from research teams located at diverse locations across the world, including North America, Europe and Asia.

In this survey, we provide a holistic coverage of the RQP innovations, and highlight their strengths and limitations. Further, we enumerate a set of open technical problems and research directions that need to be studied to make RQP a contemporary reality.

1.1 Overview of Contents

The definition of robustness has itself been a subject of intense debate for a long time, and a consensus view has been difficult to achieve (Graefe *et al.*, 2010). For instance, if worst-case performance is improved at the expense of average-case performance, is that an acceptable notion of robustness? Or, would graceful degradation, as opposed to "performance cliffs", be the right perspective? Alternatively, is it the ability to seamlessly scale with workload complexity, database size and distributional skew? As yet another option, could we settle for providing strong theoretical guarantees relative to the oracular ideal? Perhaps, the real answer is that robustness encompasses all of these scenarios and more, with the specific choice being application-dependent.

The above semantic tangle is further complicated by the different levels at which notions of robustness can be introduced – for instance, at the granularity of individual *operators* (e.g. Borovica-Gajic *et al.*, 2018), or through entire query *plans* (e.g. Chu *et al.*, 1999), or over end-to-end query *executions* (e.g. Dutt and Haritsa, 2016). Moreover, one can take *algorithmic* (e.g. Tzoumas *et al.*, 2013), *statistical* (e.g. Wu

et al., 2013b) or *learning-based* (e.g. Malik *et al.*, 2007) approaches to incorporate the robustness features at individual levels.

In this monograph, we cover representative techniques along these various dimensions. Specifically, the survey is organized in the following sequence of sections:

Section 2: Background to Robust Query Processing We begin with an overview of declarative query optimization and processing. Then, we motivate the need for RQP and the systemic challenges faced in addressing this need. In particular, we focus on the two statistical models that fundamentally underlie the optimization process, namely, *operator cardinality estimation* and *operator cost estimation*. These models address orthogonal aspects of the data processing environment – the cardinality model captures the distributions and correlations present in the data, whereas the cost model reflects the behavior of the underlying hardware and physical operator implementations.

Section 3: Robust Operators Here, we consider robustness at the granularity of individual operators, with primary focus on the Scan and Join operations which carry out much of the heavy lifting in answering user queries. The basic idea is to design adaptive or unified operators that provide close to the best performance under all execution scenarios. Such an operator would make it unnecessary for the optimizer to choose between alternatives, thereby, by definition, eliminating erroneous choices.

Section 4: Robust Plans The next stage covers entire query plans, where we consider both strategies that are robust in expectation over a workload, and those that provide robustness on an individual query basis. The latter techniques leverage simple geometric assumptions on the behavior of *plan cost functions* – for instance, monotonicity with respect to predicate selectivity. We also look into how robustness can be effectively achieved by replacing the supposedly optimal plan with a mildly sub-optimal but stable alternative.

Section 5: Robust Query Execution We then move up from optimization to robust execution of entire queries. A key feature here is that the performance metrics are in comparison with the (offline) ideal – that is, the *lower bound*. This is a major conceptual shift from the norm in the earlier literature, where the comparison was always with the "upper bound", that is, the best among the alternative competing strategies available at the time.

Section 6: Structural Robustness Bounds The bounds in the previous section are dependent on the behavior of the database query optimizer over the parameter space. Here, we provide bounds that only depend on the *structure* of the parameter space, and not its *contents*. This quantum jump in robustness is achieved through the use of "spilling", wherein the outputs of intermediate operators in the plan tree are dropped on the ground, and not forwarded to the downstream nodes. The techniques presented here continue to leverage geometric assumptions on plan cost function behavior – specifically, in addition to monotonicity, both concavity and axis alignment are considered.

Section 7: Robust Cost Models While the operator cardinality model is the primary culprit for poor query performance, robustness can also be adversely impacted by errors in the operator cost model – we discuss mechanisms for addressing this problem in this section. In particular, we focus on how statistical approaches, augmented with careful calibration and focused sampling, can perform as well or even better than learning-based approaches, while being significantly more efficient wrt both training and inference.

Section 8: Machine Learning-based Techniques Learning-based approaches to RQP, which have been hotly pursued in recent years, are discussed here, covering both query-based and data-based techniques. The former is an example of supervised learning, with models constructed by training on a large set of queries and leveraging the actual cardinalities observed during execution as the labels. On the other hand, the data-based techniques fall under unsupervised learning, and model the joint probability density functions of the underlying data to capture distributions

and correlations. Finally, there are hybrid models that leverage both queries and data in their learning process.

Section 9: Holistic Robustness Here, we show how the techniques discussed in the previous sections, which are at different layers in the database architecture, could be cohesively brought together in a complementary manner to maximize the overall system robustness.

Section 10: Future Research Directions Finally, we conclude with a suite of open research problems and future directions. The issues we pose include the impact of join-graph structure on robustness, creation of robustness benchmarks, and invoking ML techniques to determine when to use robust alternatives as opposed to their current avatars in database engines.

Overall, the big picture is that a rich variety of possibilities are currently available, and a judicious selection among them could lead to the desired robustness. Moreover, with the advent of the so-called Big Data world, wherein data is the engine driving virtually all aspects of human endeavor, the role of RQP assumes critical proportions.

1.2 Target Audience

Robust support for declarative query processing has been a long-standing concern for the database community, so we expect this monograph to be of broad relevance. In particular, the target audience for this monograph includes researchers, developers and students with an interest in the internals of database engines. The background expected is that of an introductory database systems course covering relational data models, declarative query languages, and basic query optimization and processing techniques.

Database researchers can expect the survey to provide fresh and radical perspectives on a classical research topic, serving to stimulate work on the further development of stable and efficient database engines. From the perspective of system developers and practitioners, the concepts and techniques presented in the monograph can serve as potent

mechanisms for the redesign of their systems. Finally, for database instructors and students, the coverage will help in comprehending and appreciating the complexities and subtleties of industrial-strength query processing, going far beyond the toy examples typically covered in a classroom setting.

In particular, it may influence nascent PhD students looking for challenging research topics to cast their net beyond the middleware topics occupying center stage today – this is particularly important since the benefits of new engine technologies are automatically bestowed on all applications running on these platforms.

The primary source material for the monograph consists of the papers discussed in the various sections, complemented by supporting inputs from the rich corpus of literature on query optimization and processing. A sampling of relevant publications is given in the reference list, with emphasis on recent contributions to the field.

2

Background to Robust Query Processing

A distinctive USP of SQL, which accounts for its enormous popularity, is its *declarative* nature. That is, the user or programmer only has to specify the *end* goals and not the *means* to achieve these goals. For instance, the SQL query shown below produces, on a university database, the list of students and the courses for which they are registered:

```
select S.Name, C.Title
from STUDENT S, COURSE C, REGISTER R
where S.RollNo = R.RollNo and R.CourseNo = C.CourseNo
```

What is left unspecified in this formulation is the order in which the joins $(S \bowtie R)$ and $(R \bowtie C)$ should be executed. The combined result of these joins would be the same regardless of the order, since the Join operator is both commutative and associative. However, the computation *times* could be vastly different for alternative sequences. Further, even if the appropriate sequence is known apriori, what is still left unspecified is the *physical* implementation of these logical joins, for which a variety of alternatives – *Nested Loops*, *Sort Merge*, *Hash Join*, *Index Join*, etc. – are available (Silberschatz *et al.*, 2020). Finally, apart from Join, there are a host of other operators, such as filter-predicate evaluation, for which similar decisions need to be made in enterprise queries. It is the

responsibility of the *query optimizer* module, located within the database engine, to make these decisions and come up with a recommended query execution plan.

2.1 Query Execution Plans

An execution plan is typically a tree of operators with data flowing from the leaves, representing the user tables, to the root, where the final result is materialized. A sample plan is shown in Figure 2.1 – here, a B-tree index on `CourseNo` is used to scan the COURSE table, producing output that is sorted on this attribute. Concurrently, there is a sequential scan of the REGISTER table, which is then explicitly sorted on `CourseNo`, leading to a merge join between the COURSE and REGISTER tables. The next step is a hash join between the STUDENT table and the outcome of the first join. So, the join sequence is $(S \bowtie_{HJ} (C \bowtie_{MJ} R))$. And then the final step is to return the results to the user.

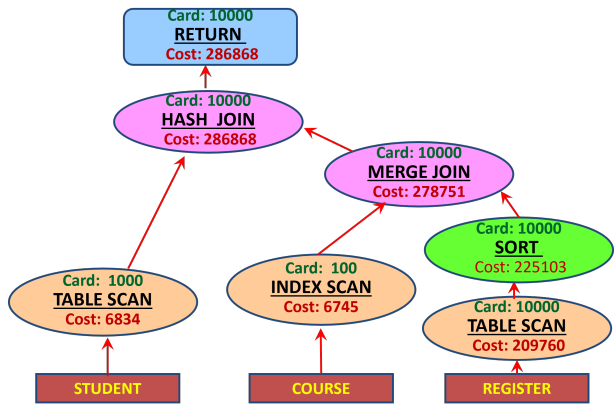


Figure 2.1: Query Execution Plan Example.

Each of the plan operators is annotated with two numbers – `Card`, shown in green, and `Cost`, shown in red. `Card`, which is short for cardinality, represents the estimated number of output rows from this operator. Essentially, it captures the volume of data flowing downstream from one operator to the next in the plan tree. For instance, the output of the index scan on COURSE is 100, which means that we expect 100 rows from this table to be piped into the upcoming join with REGISTER.

On the other hand, `Cost` is the estimated time (in abstract units) that an operator would take to complete the processing of its inputs. The costs are usually shown in an aggregate fashion, capturing the total cost of the *subtree* rooted at the operator. So, as we go up the tree, the numbers keep increasing in magnitude. As a case in point, the cost of the complete subtree rooted at the hash join is estimated to take 286868 time units. This is also the time to complete the query since the hash-join is the final computing step.

2.2 Plan Selection Process

Given a query, there is usually an exponential number of alternative plans to execute the query. For instance, just taking join ordering into account, a query with N relations has, in principle, $N!$ alternative join sequences that could be considered for evaluation. When additional choices such as join algorithms and table scan mechanisms are added, the search space explodes rapidly. So, how does the query optimizer figure out the optimal (i.e. minimum total `Cost`) plan in this ocean of possibilities?

The plan selection methodology is captured in Figure 2.2. We see here that a dynamic-programming (DP) technique is invoked to efficiently navigate through the search space. While the inherent exponential complexity remains, the DP approach reduces the search overheads to the extent that queries with up to about a dozen relations can be exhaustively evaluated on contemporary hardware, and typical decision-support queries are usually within this table cardinality threshold. For queries that exceed this table cardinality, there are heuristic techniques based on simulated annealing or machine learning that can be leveraged to identify a reasonably good plan (e.g. Swami, 1989a). The full details of the optimization process are available in standard database textbooks (e.g. Silberschatz *et al.*, 2020), so we will not repeat them here. Finally, given our assumption that `Cost` refers to response time, the output of the optimizer is the *fastest plan* to execute the user query – we assume this objective in the rest of the monograph. Of course, there are alternative metrics, such as latency or resource consumption, that could instead be used as the selection criterion.

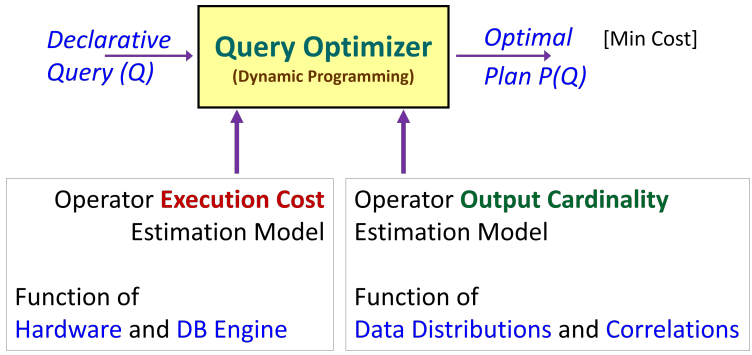


Figure 2.2: Query Optimization Framework.

There are a pair of models that serve as estimators to guide the database query optimizer in its comparison of alternative plans. The first is the *execution cost* estimation model, which estimates the red `Cost` numbers. Its predictions are primarily a function of: (a) The quality of the hardware platform on which the database engine is hosted – for example, whether the persistent data is stored on a mechanical hard disk, or a SCSI disk, or a solid-state disk, and so on; and (b) how well the software has been written within the database engine.

The second model is *cardinality estimation*, which computes the green `Card` numbers for the plan operators. These numbers are logical values that are essentially a function of the data distributions – specifically, how the data is distributed within a column, as well as the correlations across columns within a table or between tables.

2.3 Optimization Woes

At first glance, the above methodology looks perfect for producing high-quality plans for user queries. However, in practice, the reality is markedly different – the supposedly optimal plan choice at compile time may actually turn out to be highly suboptimal at runtime, even extending to *orders of magnitude* degradation!

The reason for this adverse and unexpected situation is that there are substantive *errors* in both estimation models. For the `Card` model, the errors arise due to a variety of well-documented technical reasons,

including coarse statistics, outdated metadata, attribute value independence (AVI) assumptions, multiplicative error propagation through the plan tree, and the query construction format. Whereas, for the `Cost` model, the errors arise due to simplistic linear models, operator-agnostic features, global "one-size-fits-all" model coefficients, and system dynamics. We will not describe these error-generating triggers in detail here since our goal in RQP is to work *around them* – suffice it to state that they are commonplace in contemporary systems, and we therefore need to devise robust solutions that circumvent these persistent hurdles.

To forestall the obvious question as to why the triggers themselves have not been fixed, here is an admittedly contrived but pedagogically useful instance of the hard challenges involved in designing accurate models. Consider the EMPLOYEE and MANAGER tables shown in Figure 2.3, which capture the names and ages of (a billion) employees and (a million) managers. All employees are 25 years old whereas all managers are 50 years old.

EMPLOYEE

EID	Name	Age
1	DeWitt	25
2	Chaudhuri	25
3	Franklin	25
4	Ioannidis	25
5	Lohman	25
....	....	25
10^9	Whang	25
$10^9 + 1$	Jayant	50

MANAGER

MID	Name	Age
1	Musk	50
2	Bezos	50
3	Ellison	50
4	Nadella	50
....	....	50
10^6	Altman	50

Figure 2.3: Volatility of Join Outputs.

Observe that the output of the query *"List names of all employee-manager pairs with a common age"* – i.e. computing the join $(E.Age \bowtie M.Age)$ – would be zero since there are no common ages between managers and employees. However, if we added just a *single employee (Jayant) with age 50* to the EMPLOYEE table, the join output would

immediately jump to a *million rows*! It is virtually impossible for any summarization mechanism, whether based on statistical techniques or learning methods, to capture such "nanoscopic" changes. The essential problem here is that cardinality estimation can be *hypersensitive* to data distributions – that is, even miniscule changes in the inputs may result in huge impacts on the output.

2.4 Prior Approaches to RQP

We saw above that RQP is a hard problem from both design and implementation perspectives. Given its long-standing existence and high relevance to operational behavior, a rich body of literature has developed on addressing these challenges (for instance, see Harmouch and Naumann, 2017, for a survey on cardinality estimation). A sampling of representative techniques from this prior corpus is summarized below. In this discussion and the rest of the monograph, we will use both the terms cardinality and *selectivity*, which are used interchangeably in the literature. Selectivity is the normalized [0,1] version of cardinality, the normalization being done with respect to the maximum possible output cardinality.

2.4.1 Mathematical Frameworks

An intellectually appealing approach is to come up with more sophisticated estimation frameworks – for instance, instead of the simple single-column equi-depth histograms used in current systems to model data distributions, to build wavelet-based histograms (Matias *et al.*, 1998), or self-tuning histograms (Aboulnaga and Chaudhuri, 1999), or multidimensional histograms (Muralikrishna and DeWitt, 1988). However, creating and maintaining such structures incurs considerable storage and computational overheads, and they therefore have not achieved viability.

2.4.2 Stable Plans

A more practical alternative approach is to identify stable plans that work well over large parts of the parameter space. Here, instead of

trying to make a better estimate, the goal is to preferentially pick plans expected to be robust (in terms of their performance relative to the optimal) to the estimation errors for the given scenario.

For example, in the Least Expected Cost (LEC) approach (Chu *et al.*, 1999; Chu *et al.*, 2002), it is assumed that the distribution of predicate selectivities is apriori available, and the plan with the least-expected-cost over the distribution is chosen for execution. While this approach is good for improving the average case, it may still suffer poor performance in the worst-case. Further, it assumes prior knowledge of query distributions, which may be difficult to obtain in practice, especially in dynamic environments.

An alternative Robust Cardinality Estimation (RCE) strategy proposed in Babcock and Chaudhuri (2005) is to model the cardinality dependency of the cost functions of the various competing plan choices. Then, given a user-specified "confidence threshold" T, the plan that is expected to have the least upper bound with respect to cost in T percentile of the queries, is selected as the preferred choice. The choice of T determines the level of risk that the user is willing to accept with regard to worst-case behavior. Like the LEC approach, this strategy can also be arbitrarily poor for a specific query, as compared to the optimizer's choice.

The above techniques minimize the *expected* cost. In contrast, the *variance* of the plan quality was used as the optimization metric in Chaudhuri *et al.* (2010), and low-variance plans were preferentially chosen for execution.

2.4.3 Plan Switching

Another approach is to use the classical optimization technique as is at compile-time, but then, as the query progresses in its execution, to keep comparing what was expected with what is actually encountered. If the two values are in the same ballpark, the execution is continued. However, if a big discrepancy arises – for example, the join is estimated to produce 10 tuples, but the output turns out to be 100000 rows instead, or the other way around – we return to the optimization drawing board, and redo the optimization process. The difference this time around, however,

is that some of the estimated values are replaced with those that have been observed during the partial execution. The updated plan based on these inputs is taken up for execution, and monitored again for any subsequent deviations. This "trial-and-error" movement towards a good plan continues until the query completes its full processing.

Techniques based on the above paradigm are termed *plan-switching approaches*, as they involve run-time switching among complete query plans. Well-known exemplars include Dynamic Re-optimization (Kabra and DeWitt, 1998b), POP (Markl *et al.*, 2004) and Rio (Babu *et al.*, 2005). However, these techniques also employ heuristics that do not lend to quantifiable robustness guarantees. For instance, POP may get stuck with a poor plan since its selectivity validity ranges are defined using structure-equivalent plans only. Similarly, Rio's sampling-based operators for monitoring selectivities may not work well for join-selectivities. Further, it assumes that if the plans chosen by the optimizer at the corners of the principal diagonal of an axis-parallel box in the multi-dimensional selectivity space are the same as those chosen at a point estimate within the box, then this plan is robust throughout the box. However, this assumption is often violated in practice, especially for large boxes.

2.4.4 Multi-plan Execution

A radically different query execution approach was proposed in Eddies (Avnur and Hellerstein, 2000), wherein, in principle, each tuple within a query could have a distinct plan – that is, multiple concurrent plans, as opposed to the sequential execution in the plan-switching approach. Leveraging this idea, dynamic routing of individual tuples was advocated for robustness in Neumann and Galindo-Legaria (2013). Here, multiple plans proceed in parallel, and through a system of "back pressure", tuples are preferentially led away from inefficient plans towards efficient alternatives. Further, the execution is always progressing forward. On the flip side, this approach poses considerable difficulties from both implementation and repeatability perspectives.

2.4.5 Execution Feedback

Another unique twist to achieve robustness was presented in Zhu *et al.* (2017) for the restricted context of pipelined left-deep hash-join trees operating on in-memory star schema data warehouses. The core idea here is to neutralize the impact of poor join-order permutations through appropriate query *execution* strategies at run-time. That is, to drastically reduce the inherent variance in plan costs such that the optimizer's plan selection becomes effectively immaterial. This cost conflation is achieved through "Lookahead Information Passing" (LIP), where information about downstream joins is propagated to the earlier joins. The advance information helps reduce the redundant hash probes for tuples destined to be filtered out by later joins, and thereby minimizes their adverse impact on performance. An additional benefit of LIP is that its conflated costs may be even lower than that of the native optimal join order! Finally, the implementation is minimally invasive wrt to code changes in the optimizer and execution modules. However, extending these ideas from their highly limited context to generic database environments – for instance, bushy plan trees or other relational operators – remains a challenging open problem.

2.4.6 Multi-relation Joins

An article of faith in the database community, especially among practitioners, is that joins are *binary* operators, combining a pair of relations at a time. However, this established canon was recently overturned in the so-called "worst-case optimal joins" (WCOJ) stream of work (Ngo, 2018). The core idea, as exemplified by Generic Join (Ngo *et al.*, 2014), is that the join processes one attribute at a time but simultaneously combines *all relations* sharing that attribute. The WCOJ approach leads to algorithms whose run-times provably match the worst-case output size of a given join query. So, they are asymptotically faster than binary joins and, therefore, more robust to sub-optimal plan choices. However, they can be arbitrarily worse than binary joins for specific database and query instances. Finally, there have also been hybrid proposals, such as Free Join (Wang *et al.*, 2023), that aim to integrate WCOJ and binary joins in a synergistic manner that combines their individual benefits.

2.4.7 Multi-resource Execution

A common source of large performance cliffs is execution spilling over from a primary hardware resource to a secondary device – for instance, from in-memory processing to disk-resident processing. The impact of the cliff can be reduced by overlapping different parts of the computation across these platform hierarchies. A recent example of this strategy is a dynamic hybrid hash join that smoothly adapts its policies and parameters to relational inputs that are uncertain with regard to size and distribution, as well as record lengths and storage technologies (Jahangiri *et al.*, 2022).

2.5 Summary

The above-mentioned techniques feature a variety of novel ideas and formulations. However, and here lies the crux of the problem, they are all essentially *heuristic* techniques – while potentially working well in some scenarios (typically the average case), they are unable to promise strong numerical robustness guarantees in a universal sense (typically failing in the worst case), and individual queries may suffer arbitrary degradation. This is particularly problematic given that, unlike most computing applications, where the average case is typically weighted towards the best case, the unfortunate situation in database environments is that the average case is often closer to the worst case, as highlighted in Figure 2.4.

Figure 2.4: Typical Performance in Database Environments.

However, as mentioned in the Introduction, there have been recent research advances that collectively promise to provide robust foundations for designing the next generation of query processing engines. These new techniques are presented in the remainder of this monograph.

Our focus here is exclusively on exact query processing since many applications require precise results. However, there is also a significant

body of work that aims to provide performance robustness via *approximate* query processing – a recent survey on these techniques, covering both sampling-based and learning-based strategies, is available in Li and Li (2018).

3

Robust Operators

We begin our study of robustness at the *operator* granularity. The core idea here is simple – design operator implementations that are the *best*, or close to the best, under *all* query scenarios. Achieving this objective automatically eliminates the possibility of making an incorrect choice between alternative competing implementations. The challenge, of course, is to design such "universal" techniques which provide the best of all worlds. We cover two such exemplars here – `SmoothScan` (Borovica-Gajic *et al.*, 2018), which adaptively switches between sequential and index scans, and `G-Join` (Graefe, 2012), which unifies the design of index nested-loops, sort-merge and hash joins.

3.1 Smooth Scan

The design methodology underlying `SmoothScan` is to *dynamically* morph between sequential full scan (FS) and index scan (IS), based on the observed statistical properties of the input data. Specifically, at small selectivity values, it behaves similar to IS, whereas for larger selectivities, it progressively changes its behavior towards FS.

To motivate this design, consider the performance graph shown in Figure 3.1. The X-axis refers to the TPC-H benchmark (Transaction

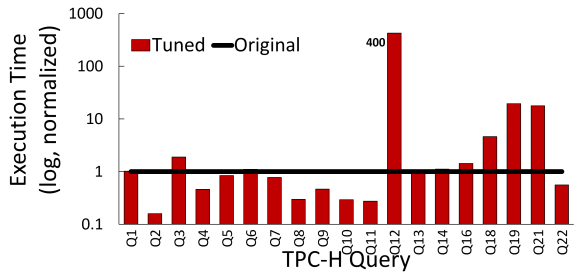


Figure 3.1: Impact of Index Tuning (Borovica-Gajic *et al.*, 2018).

Processing Council, 2024b) queries, Q1 through Q22, on a scale factor 10 database. The queries were run on a commercial database system, referred to as COM, in two environments – the first with the default configuration, and the second populated with indexes proposed by the tuning tool of COM. On the Y-axis is the response time of the tuned version normalized to the default, on a log-scale.

What we see in the figure is that although the performance of some queries does get significantly improved (e.g. Q2) thanks to the index tuning, there are also quite a few where the performance markedly *deteriorates*. As a particularly egregious example, the execution of Q12 slows down by a factor of 400! This surprising degradation is due to the query optimizer choosing the wrong scan – IS over FS, an outcome of severely underestimating the underlying predicate selectivity.

More fundamentally, in conventional databases, the choice between IS and FS is a hard-wired *binary* decision, committed to at the beginning of the scan. Further, the cost function of IS is essentially linear in the selectivity, whereas that of FS is a constant, independent of selectivity, as shown in Figure 3.2. Now, even if we did realize along the way that a mistake had been made in the choice of scan, switching to the right choice could entail a substantial transition cost, since the earlier work may have to be discarded. This is because it is not feasible, without considerable bookkeeping, to on-the-fly switch from IS to FS or vice versa. Therefore, a runtime change of strategy can result in a performance cliff when the prior investment is lost.

Ideally, we desire a scan that smoothly transitions between IS and FS based on the underlying data, as shown by the green line in Figure 3.2.

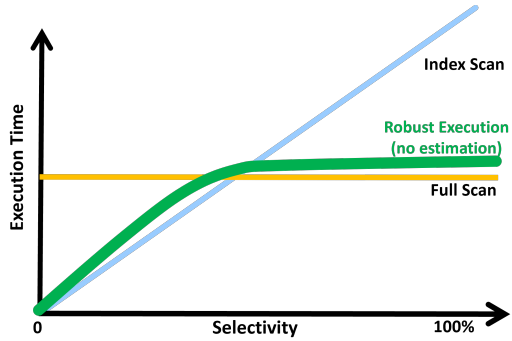


Figure 3.2: Performance Profiles of Scan Operators (Borovica-Gajic *et al.*, 2018).

And, to do so based on the encountered data characteristics, and not on hardwired estimation formulas. In essence, a *data-driven* approach instead of *estimation-driven* choices.

3.1.1 Morphing Modes

`SmoothScan` achieves adaptivity by morphing between the three modes of access shown in Figure 3.3. The first is Mode 1, corresponding to the traditional index access. The second, Mode 2, leverages the fact that although the traditional index is used to access a specific tuple in a page, it brings the entire page into memory. Therefore, all other tuples in the page can be proactively checked for a "free" predicate match. The last mode, Mode 3, takes this piggy-backing idea even further – it probes not just the current page, but also its close neighbors, which are cheap to retrieve since the disk head is already close to their location, essentially "flattening" the scan.

The move from one morphing mode to another is achieved through a simple feedback-based elastic process. That is, when there is a selectivity increase, the mode level is increased since the desirable choice is towards FS. Analogously, when there is a selectivity decrease, the mode level is decreased, reverting towards IS. Further, in Mode 3, the size of the neighborhood is progressively increased or decreased based on the fraction of *productive* pages (i.e. pages where one or more results are available) retrieved in the latest flattening access, as compared to

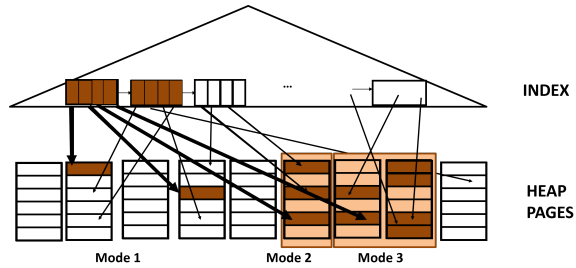


Figure 3.3: `SmoothScan` Modes of Operation (Borovica-Gajic *et al.*, 2018).

the overall productivity evaluated over the entire index. The expansion/reduction function is a *doubling* algorithm, which allows for quick adaptation to distributional changes without becoming unduly reactive to temporary spikes.

3.1.2 Performance Profile

How does `SmoothScan` work in practice? Sample results on `PostgreSQL` for the motivating experiment described earlier are shown in Figure 3.4. We observe here that for large selectivities, `PostgreSQL` usually makes the right choice. On the other hand `SmoothScan`, despite starting with IS, quickly adjusts to Mode 3 and is only marginally worse. And when we consider low selectivity values, since they are on large tables (specifically, LINEITEM which has 10M tuples), the absolute number of tuples retrieved is still large enough that FS is preferable. However, `PostgreSQL` chooses IS due to underestimating the selectivity by about two orders of magnitude, resulting in poor performance. Whereas `SmoothScan` again provides adaptive performance that is about an order of magnitude better.

Robustness Guarantees

More importantly from a robustness perspective, it was shown in Borovica-Gajic *et al.* (2018) that `SmoothScan` lends itself to providing *quantitative* guarantees wrt the optimal – specifically,

$$\frac{\texttt{SmoothScan}}{Ideal} \leq (1 + \frac{rand\_io\_cost}{seq\_io\_cost}) \tag{3.1}$$

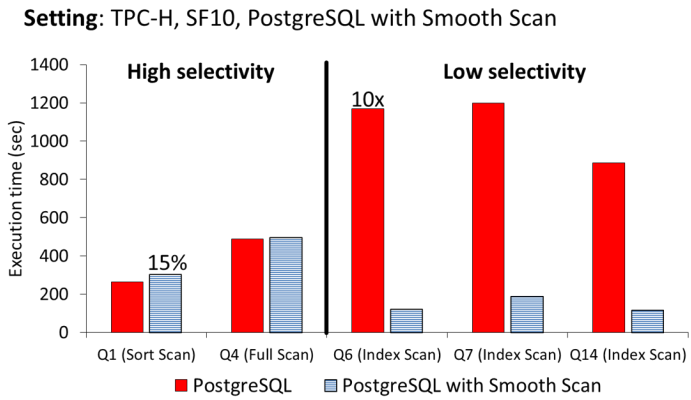


Figure 3.4: Performance of `SmoothScan` (Borovica-Gajic *et al.*, 2018).

where *rand_io_cost* and *seq_io_cost* are the relative costs of random disk access and sequential transfer. Here, the Ideal is represented by an oracular algorithm where only the relevant pages of the table are sequentially read from disk. For representative contemporary HDD parameters, the sub-optimality factor of `SmoothScan` is around 11, whereas for SSD disks, it comes down to 6.

3.1.3 Book-keeping Structures

While the basic idea of dynamic morphing is attractive, it comes at a considerable price in maintenance. Specifically, to ensure that the result semantics, such as duplicates and ordering, are maintained despite the dynamic mode switches, additional book-keeping data structures are required. These include a *Page ID* cache to ensure a given data page is not processed multiple times, a *Tuple ID* cache to ensure a given result tuple is not produced multiple times, and a *Result Cache* to ensure ordered output. Further, since the sizes of these caches are variable, memory management techniques need to be incorporated to handle changing demands.

Overall, `SmoothScan` provides robust behavior without requiring accurate prior statistics. It brings significant gains when the original system makes a wrong decision, and incurs only marginal overheads when a correct decision is made. However, on the down side, it requires

a significantly invasive change of the system internals to support its implementation.

3.2 Generalized Join

A second unified operator is `G-Join` (Graefe, 2012), which fuses the popular join algorithms (*Index Nested Loops*, *Sort Merge* and *Hash*) into a common framework. Note that unlike `SmoothScan` which dynamically morphs between modes, `G-Join` is an umbrella algorithm.

We begin by showing how Sort Merge can be implemented using concepts from Hash Join. If the inputs are already sorted, then just execute the standard Sort Merge which is optimal for such scenarios. On the other hand, if they are not presorted, create internally sorted runs, using replacement selection, for both inputs – however, do not carry out the merging steps. Instead, similar to hash partitions, store "key-covering pages" from the smaller input (R) in a buffer pool, and assign a single buffer page for the larger input (S). What is meant by key-covering is the set of pages in R containing tuples that cover the key-range of the tuples in the buffer page of S. The intention here is that each page of S should be brought into memory only once. Therefore, the R buffer pool is dynamically expanded until it key-covers the buffer page of S, and then these memory-resident pages are joined. After this operation is completed, the next S page is brought into memory, and the process repeats. The R buffer pool is dynamically shrunk if any page goes outside the key coverage range.

The above algorithm is shown in graphical form in Figures 3.5 and 3.6. In the first "build" phase, similar to Hybrid Hash Join, hash partition R_0 is constructed in memory. The difference is that instead of using the remaining memory as partition output buffers, they are used for *sorted run generation* – $R_1, \ldots, R_K$ – which are written out to disk. A cut point is used to partition the input so that tuples with join keys less than the cut point are used to construct R_0, while the others are assigned to run generation.

In the second "probe" phase, some of the runs of R are read, with the choice guided by the priority queue A, which keeps track of the range of values covered by the pages of each run. The priority queue B keeps track of which pages can be dropped from R's buffer pool.

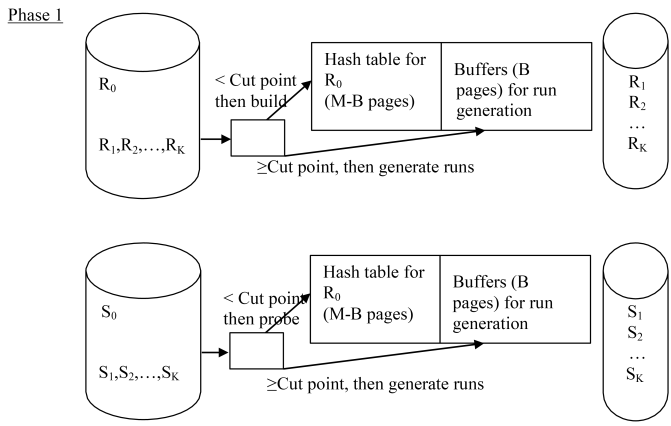


Figure 3.5: `G-Join` Phase 1 ("Build") (Li, 2010).

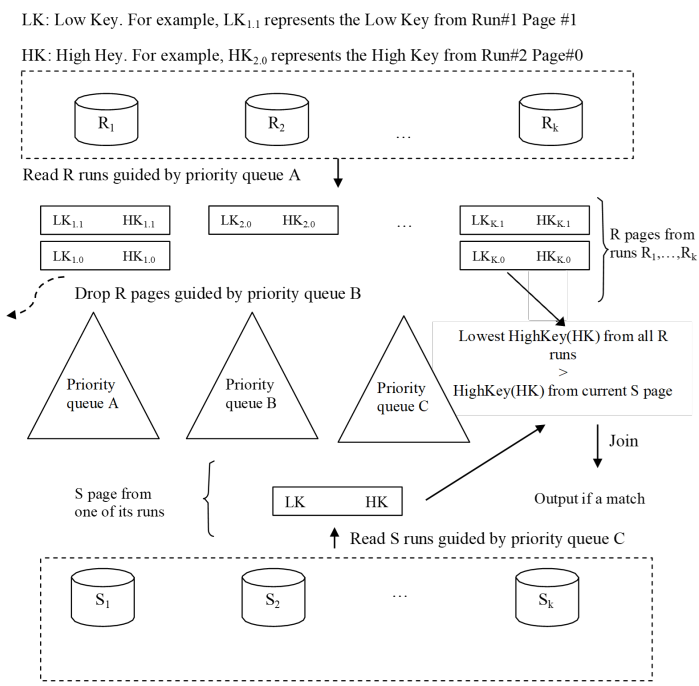


Figure 3.6: `G-Join` Phase 2 ("Probe") (Li, 2010).

And finally, the priority queue C is used to access the pages of S, one at a time, from its runs. In the above, the choice of sort algorithm, *replacement selection*, is deliberate since it allows creating runs that are on average *double* the size of memory, whereas standard algorithms such as quicksort only provide runs that are commensurate with memory size.

While the fusion of Sort Merge and Hash Join was discussed above, we now consider the correspondence to Index Nested Loops join. Here, the expectation is that there is an index on the large inner input (S) and that there are relatively small number of distinct join key values in the outer input (R), which means that reading useless inner pages should be avoided. For this scenario, `G-Join` sorts the small input, and then performs a *zigzag* merge join of the two inputs – that is, the merge attempts to skip over useless input records and applies this logic in both directions between the join inputs. If the number of distinct join key values in the outer table is small, then many of the pages in the inner input are never needed for join processing. This is, of course, precisely the effect and performance advantage of Index Nested Loops join.

3.2.1 Performance Profile

A sample scaling performance with database size of the `G-Join` (GJ) algorithm, relative to Hash Join (HJ) and Merge Join (MJ), is shown in Figures 3.7 and 3.8 for sorted and unsorted inputs, respectively. We see here that with presorted inputs, `G-Join` is only marginally worse than the optimal, Merge Join. And with unsorted inputs, it is only marginally worse than Hash Join, the de facto leader in these environments. So, again, we essentially have the best of both worlds. And, what is more important, the possibility of optimizer errors in making algorithmic choices has now disappeared due to the unified implementation.

While `G-Join` may work well in general, it may encounter difficulties if there is a heavy skew in either the sizes of the runs or the key value ranges. For instance, if a given page of S has a very wide key range, then the entire R has to be brought sequentially into memory in order to complete the join.

Finally, similar unifications have been developed for the grouping, aggregation and duplicate elimination operators as well in Graefe (2012).

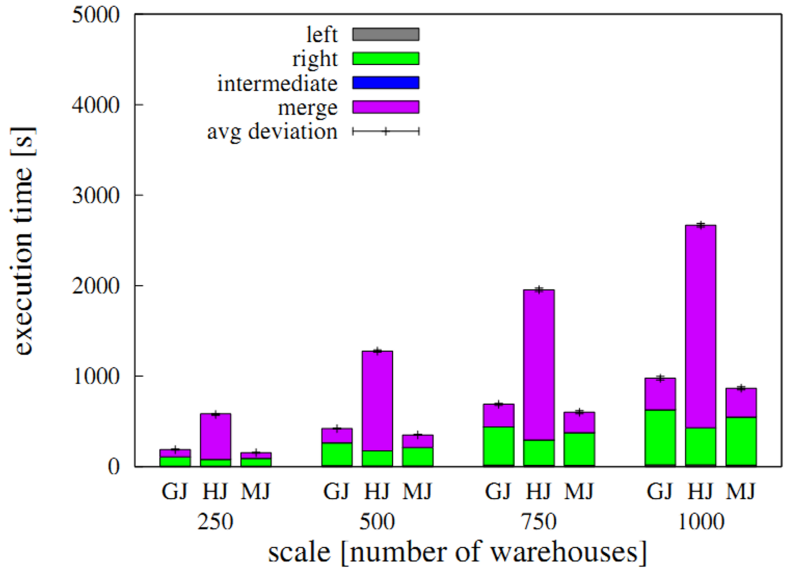


Figure 3.7: `G-Join` Performance – Sorted Inputs (Graefe, 2012).

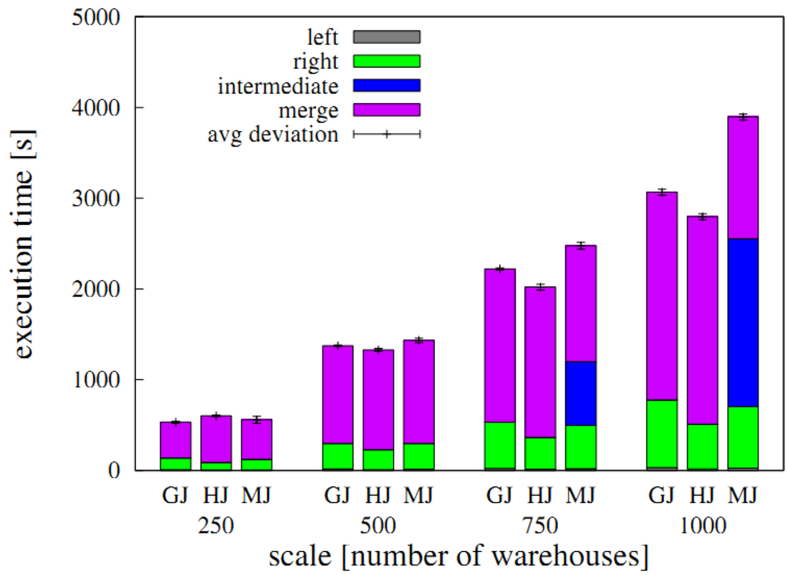


Figure 3.8: `G-Join` Performance – Unsorted Inputs (Graefe, 2012).

4

Robust Plans

We next consider techniques that aim to provide robustness at the granularity of entire plans. Contemporary optimizers typically approximate the distributions of run-time parameters with representative values – for example, the mean or mode – and then choose the corresponding plan to execute all instances of the query. But this will obviously not work well if the actual values encountered at run-time are significantly different from the representative values. Therefore, as previously introduced in Section 2, an alternative strategy proposed in Chu *et al.* (1999) is instead to optimize for the "least expected cost" **(LEC)** plan, where the expectation is computed over the full distribution of the input parameters.

However, there are practical limitations with this approach: First, determining the LEC plan involves substantial computational overheads when the number of plans over the parameter space is large. Second, it assumes that the candidate plans have all been modeled at the same level of accuracy, rarely true in practice. Third, and most importantly, the robustness here is with respect to entire workloads, and not with regard to individual queries, which is what we would ideally like to have.

In this section, we present two techniques, `CostGreedy` (Doraiswamy *et al.*, 2007) and `SEER` (Doraiswamy *et al.*, 2008), that provide robustness at individual query granularity. To set the stage for their description, we begin by describing the concept of "plan diagrams" introduced in Reddy and Haritsa (2005).

4.1 Plan Diagrams

A plan diagram is a color-coded pictorial enumeration of the plan choices of the optimizer for a parameterized query template over its relational selectivity space. As an exemplar, consider the query template shown in Figure 4.1, based on TPCH Q8, whose goal is to determine how the market share of Brazil has changed over a two-year period in the America region for a certain type of steel part. There are two additional filter predicates – the account balance of the supplier should be less than some constant C1, and the extended price of the lineitem entry should be less than another constant C2. So, we can view C1 and C2 as parameters that control the selectivity of the SUPPLIER and LINEITEM tables, respectively. These selectivities form a 2D space, with each point in the space representing a distinct query corresponding to a unique combination of C1 and C2. The set of queries in the space, corresponding to a given resolution, are individually presented to the database query optimizer and the recommended plans are obtained. Plans that have the same "skeleton" – that is, identical query operator trees – are assigned a common color, whereas plans with different skeletons have distinct colors.

```
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
                   r_name = 'AMERICA' and s_nationkey = n2.n_nationkey and
                  o_orderdate between '1995-01-01' and '1996-12-31' and
                  p_type = 'ECONOMY ANODIZED STEEL'
                  and s_acctbal ≤ C1 and l_extendedprice ≤ C2
       ) as all_nations
group by o_year
order by o_year
```

Figure 4.1: TPCH Q8 template.

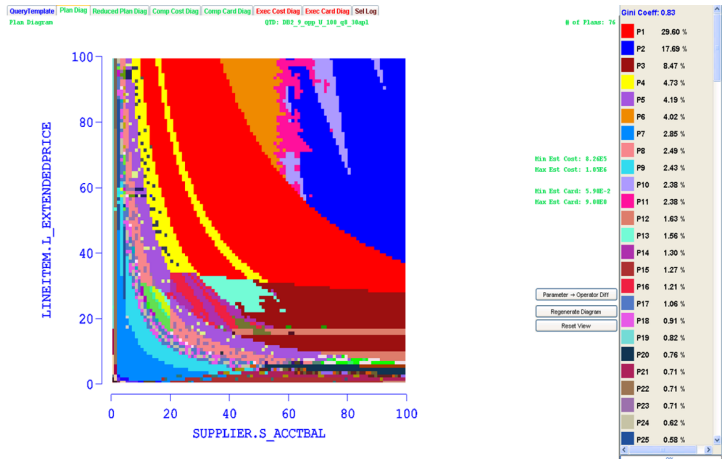


Figure 4.2: Plan Diagram for TPCH Q8 template (Doraiswamy *et al.*, 2007).

With the above procedure, the plan diagram obtained for the Q8 template, at a resolution of 100 in each dimension, is shown in Figure 4.2. Here, each colored region corresponds to a unique plan, and we see a large number of plans covering the space – 76 in all! Moreover, the geometries of the plan boundaries are highly irregular. In the legend on the right side of the diagram, the various plans, P1 through P76, are organized in descending order of area coverage. We find plan P1 occupying about 30% of the space, whereas the smallest plans occupy less than 0.1% of the space. Collectively, the plans featured in the diagram are known as the *parametric optimal set of plans* **(POSP)** (Hulgeri and Sudarshan, 2002).

4.2 Cost Greedy Reduction

In the `CostGreedy` algorithm (Doraiswamy *et al.*, 2007), the core idea is to compute the POSP over the parameter space, and then reduce it to a low-cardinality approximation, where the number of plans is much smaller. The expectation is that these retained plans are relatively stable plans with respect to estimation errors.

Specifically, the following Plan Diagram Reduction problem is addressed: *Recolor the plan diagram with the smallest set of colors (i.e. some plans are "swallowed" by others), such that no query point in the*

original diagram has its estimated cost increased, post-swallowing, by more than λ percent.

Here, λ is a user-specified parameter capturing the acceptable sub-optimality. That is, each query point in the original diagram either (a) retains its original plan, or (b) has it substituted with a replacement plan whose sub-optimality, relative to this original choice, is bounded by λ. If λ is kept small, say 10%, then there is really no material performance deterioration – this is especially so given that the suboptimalities experienced in contemporary scenarios are often in orders of magnitude.

Based on a reduction from the Set Cover problem (Garey and Johnson, 1979), it can be shown that finding the optimal reduction, wrt to the number of retained plans, is computationally hard. However, a greedy approximation that usually turns out to be close to optimal is incorporated in the `CostGreedy` algorithm. The only assumption made in its design is a simple geometric property, called Plan Cost Monotonicity, defined as follows on the *plan cost function (PCF)* of the POSP plans:

Plan Cost Monotonicity (PCM): The PCF of each POSP plan featured in the plan diagram is *monotonically increasing* over the entire selectivity space.

What this means is that given any plan P, and a location q_2 in the selectivity space, as shown in Figure 4.3, then at any location q_i in the hypercube subtended by q_2 wrt the origin, the cost of executing q_i with P will be less than that at q_i. That is,

$$Cost(P, q_i) < Cost(P, q_2) \;\; if \;\; q_i \prec q_2 \tag{4.1}$$

In a nutshell, "spatial domination implies cost domination".

Intuitively, the PCM condition captures the expectation that the cost of a plan is expected to increase with the base relation selectivities. For most query templates, this is the case since an increase in selectivity corresponds to processing a larger amount of input data. As a consequence, when considering the recoloring possibilities for a query point q_i, only those plan colors that appear in the *first quadrant*, relative to q_i as the origin, are considered. This restriction is because only a vacuous

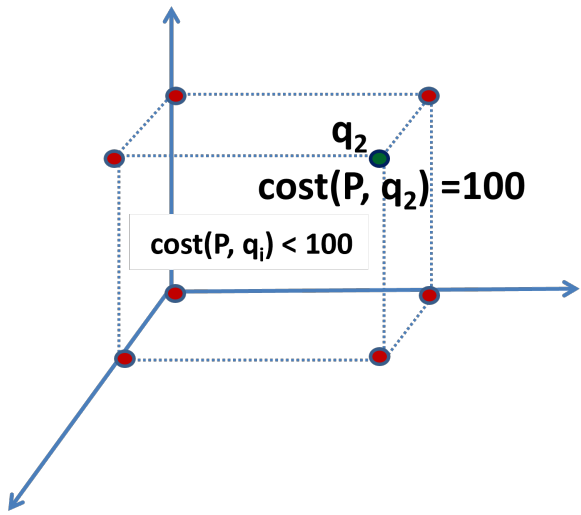


Figure 4.3: Plan Cost Monotonicity (PCM).

statement can be made about the costs of plans from other quadrants, namely that they lie in the interval $[Cost(P, q_i), \infty)$.

For the plan diagram in Figure 4.2, the `CostGreedy` algorithm first casts the diagram into a 2D grid, matching the picture's resolution. It then scans the grid from right to left, beginning with the top row of the grid, and then working its way down, row by row. This directional movement is essential since the PCM assumption can now be easily leveraged to decide when an entire plan can be swallowed by another – the details are given in Doraiswamy *et al.* (2007).

4.2.1 Anorexic Reduction

A surprising and potent outcome from an empirical evaluation of `CostGreedy` over a large number of benchmark templates, was the following: Irrespective of the number of plans in the original plan diagram, which was often in the hundreds, the final number post-reduction, even with very modest λ thresholds of 20 percent or less, came down to a single digit! In short, that plan diagrams can be reduced to "anorexic" cardinalities while retaining acceptable query processing performance.

As a case in point, the reduced picture for Q8 with $\lambda = 10\%$ is shown in Figure 4.4 – the original 76 plans are now drastically reduced to just 5 plans. Further, the geometries of the plan boundaries, which were previously highly irregular, have become smooth hyperbolic curves.

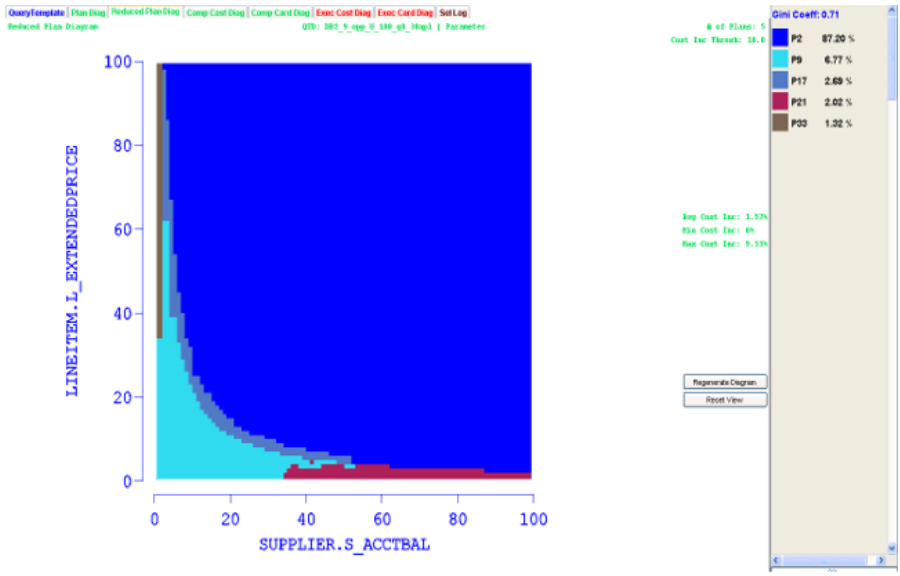


Figure 4.4: Reduced Plan Diagram for TPCH Q8 template (Doraiswamy *et al.*, 2007).

We hasten to add that anorexic reduction is not just a cosmetic exercise of reducing complex plan diagrams to simpler versions, but has several performance benefits, as outlined in Doraiswamy *et al.* (2007). Specifically, from the RQP perspective, the utility is that the retained plans are highly resistant to selectivity errors, since they are robust choices over large areas of the selectivity space. For instance, the dark blue plan in Figure 4.4 is a good choice over almost 90% of the space, therefore accounting for most of the selectivity errors that may be encountered at runtime.

Notwithstanding the above, it is still possible that `CostGreedy` may cause arbitrarily poor performance if the selectivity error is large enough that the actual location of the query falls *outside* the swallowing region of the estimated location. This scenario is captured in Figure 4.5,

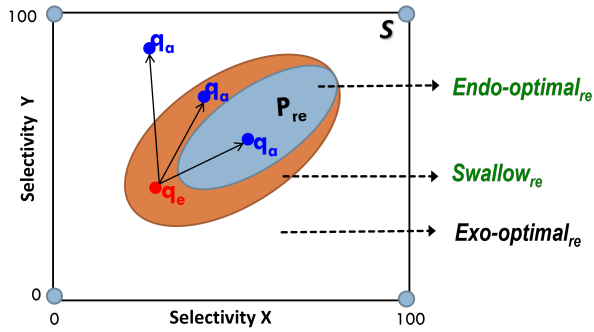


Figure 4.5: Error Location Regions wrt Plan Replacement (Doraiswamy *et al.*, 2008).

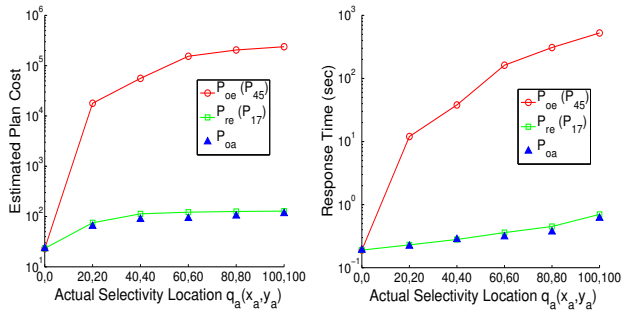
where q_e is the estimated selectivity location, q_a is the actual run-time location, P_{oe} is the optimal plan for q_e and P_{oa} is the optimal plan for q_a. Further, P_{re} is the replacement plan for P_{oe}. Here, if q_a falls in the (blue) optimality region of P_{re}, then it is in the "endo-optimal" region of P_{re}, making it inherently robust. Whereas if it is in the brown region, where P_{re} can swallow within the λ constraint, then again it is inherently robust. However, if q_a is outside these colored regions, its performance could be arbitrarily worse depending on the behavior of P_{re} at q_a.

4.2.2 Robustness via Reduction

We now present example scenarios to motivate (a) the error-resistance utility of plan diagram reduction, and (b) the need for ensuring safety in this process.

Performance Enhancement

The first scenario, shown in Figure 4.6 demonstrates how the replacement plan P_{re} can provide huge improvements *throughout the selectivity space*. Specifically, reduction was carried out with $\lambda = 10\%$ on a plan diagram for a query template based on TPC-H Q5, with selectivity variations on the CUSTOMER and SUPPLIER relations. On this diagram, for $q_e = (0.36, 0.05)$, a sample set of actual locations (q_a) along the



(a) Compile-Time (b) Run-Time

Figure 4.6: Beneficial Impact of Plan Replacement.

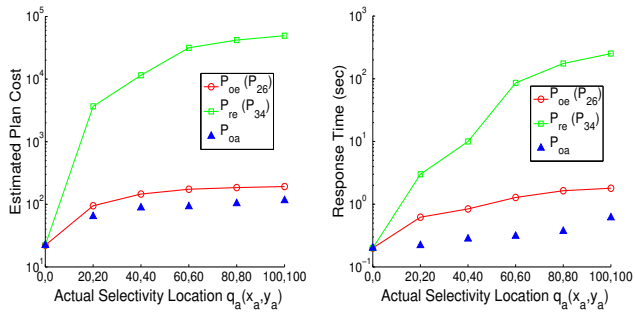
principal diagonal of the selectivity space were considered. For this instance, the costs of P_{oe} (P45), P_{re} (P17) and P_{oa} (the optimal plan at each q_a location) are shown in Figure 4.6(a) – note that the costs are on a log scale.

It is clear from Figure 4.6(a) that the replacement plan P17 provides *orders-of-magnitude* benefit w.r.t. P45. In fact, the error-resistance is to the extent that it virtually provides "immunity" to the error since the performance of P17 is close to that of the *locally optimal plan* P_{oa} throughout the space. Moreover, this sustained improvement is obtained despite the endo-optimal region of P_{re} constituting only a very small fraction of this space.

Moreover, the benefits expected from the compile-time analysis do translate to corresponding improvements at *runtime*. This is shown in Figure 4.6(b), where the query response times (again measured on a log scale) of P45, P17 and P_{oa} at the same q_a locations are presented. It is vividly clear in this picture that huge savings in processing time are 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.

Performance Degradation

While performance improvements are usually the order of the day, there are also occasional situations wherein P_{re} performs *worse* than P_{oe} at q_a. A particularly egregious example, arising from the *same* plan diagram, is shown in Figure 4.7(a) 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 is corroborated in Figure 4.7(b) which shows the corresponding query response times.



(a) Compile-Time (b) Run-Time

Figure 4.7: Adverse Impact of Plan Replacement.

From the above, it is clear that we would like to have a mechanism through which one could assess whether a replacement is *globally safe* over the entire parameter space. While global safety could, in principle, be ensured by explicitly checking every location in the exo-optimal region, it would be computationally impractical. Therefore, we present below the SEER algorithm (Doraiswamy *et al.*, 2008), which efficiently ensures that the sub-optimality guarantees are maintained irrespective of the location, including the exo-optimal space.

4.3 Global Safety with SEER Reduction

Global safety is mandated by extending `CostGreedy`'s swallowing criterion of

$$\forall \text{ points q in endo-optimality region of } P_{oe}, \\ cost(P_{re}, q) \leq (1 + \lambda) cost(P_{oe}, q) \tag{4.2}$$

to the stricter constraint that

$$\forall \text{ points q in selectivity space S}, \\ cost(P_{re}, q) \leq (1 + \lambda) cost(P_{oe}, q) \tag{4.3}$$

In `SEER`, anorexic plan diagram reduction is augmented with a generalized mathematical characterization of PCF behavior over the parameter space. As an example, for a 2D selectivity space with x and y dimensions, the PCF is expressed as

$$cost(P, (x, y)) = a_1 x + a_2 y + a_3 xy + a_4 x \log x + a_5 y \log y + a_6 xy \log xy + a_7 \tag{4.4}$$

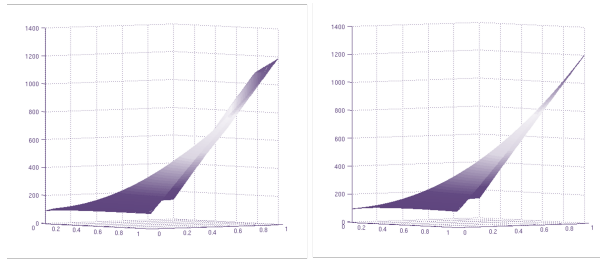
where the a_i are coefficients. The choice of terms is based on the operators typically found in query execution plans – for instance, Join is captured by the xy term, Sort by $xlogx$, Table Scan with the constant a_7, etc.

The fitting of these template functions to the specific plan functions can be carried out using standard techniques such as Linear Least Squares method (Kreyszig *et al.*, 2011). An example 2D fit is shown in Figure 4.8, where the actual cost function is on the left and the fitted function is on the right. The algebraic version of the fitted function is also shown in the figure, and it is visually evident that the fit is very good.

A **"safety function"** is defined in `SEER` as

$$safety(x, y) = cost(P_{re}, (x, y)) - (1 + \lambda) cost(P_{oe}, (x, y)) \tag{4.5}$$

which captures the differences between the costs of P_{re} and a λ-inflated version of P_{oe} in the selectivity space. A variety of safety-related checks can be constructed based on the behavior of the safety function and its first and second derivatives in this space.



$$Cost(x,y) = 17.9x + 45.9y + 1046xy - 39.5x\log x + 4.5y\log y + 27.6xy\log xy + 97.3$$

Figure 4.8: Plan Cost Functions: Actual and Fitted (Doraiswamy *et al.*, 2008).

The main result in Doraiswamy *et al.* (2008) is that passing the above safety-related checks in the *perimeter* of the selectivity space automatically implies safety in the *interior* as well – therefore, it is only required to check for safety at the border, a computationally much simpler task. In fact, it was shown that analyzing the safety function behavior at just the *corners* of the space was often sufficient to make the safety determination over the entire region.

Equally importantly, it was shown that despite having to meet a stricter replacement constraint, the reduction cardinality of `SEER` retained the anorexic profile achieved by `CostGreedy`, thereby simultaneously providing global safety and robustness.

4.3.1 Limitations

As discussed above, `SEER` essentially assures, at the granularity of individual queries, performance that is either much better than the native optimizer, or at worst, marginally suboptimal (bounded by λ percent) in comparison. Despite this strong guarantee, `SEER` still has a serious limitation – the robustness guarantees are with respect to P_{oe}, the optimal plan at the estimated location – i.e. the native optimizer's plan. That is, we are again comparing with the *upper bound* represented by the contemporary optimizers. Ideally, we would like the guarantees to be wrt P_{oa}, the optimal plan at the actual location, representing the *lower bound*. We address this issue in the next section.

5

Robust Query Execution

We now turn our attention from optimization to the robust execution of entire queries wrt the optimal. The specific performance metric used here is **Maximum Sub-Optimality (MSO)**. This metric is defined as the worst-case execution slowdown, evaluated over the entire selectivity space, relative to an oracular ideal that magically knows the correct selectivities.

5.1 Bounded Impact

An early work that attempted to provide MSO guarantees with regard to query performance was described in Moerkotte *et al.* (2009). A subtle but critical point they highlighted was with regard to the metric used to measure cardinality estimation errors. The standard metrics are either the L_2 norm, which is the Euclidean distance between the estimated and actual selectivities, or the L_∞ norm, which captures the worst-case discrepancy over all dimensions. However, the surprising result of Moerkotte *et al.* (2009) was that minimizing these error metrics could lead to arbitrarily bad plans from a general perspective.

Instead, they proposed an alternative metric called **Q-error** (where Q stands for Quotient), defined as follows:

$$q = max(\frac{EstCard}{TrueCard}, \frac{TrueCard}{EstCard}) \tag{5.1}$$

So, for instance, if the cardinality estimate for a particular relational expression is 1000 and the actual turns out to be 100, then the Q-error is 10. On the other hand, if the estimate is 1000 and the actual turns out to be 100, the Q-error continues to be 10, unlike the earlier formulas.

The Q-error metric has become the de facto standard in the query processing literature over the past decade because of the following reasons: (1) It treats underestimates and overestimates symmetrically; (2) It can be unbounded in value, whereas notions such as relative error can be upper-bounded by 1, independent of the estimated value; and (3) It matches well with the multiplicative error propagation encountered from the successive estimates made while ascending the plan tree.

The interesting outcome of the above metric is that, under certain restricted settings, bounds on Q-error translate to guaranteed limits on MSO! Specifically, the following theorem was proved in Moerkotte *et al.* (2009):

Theorem 5.1. Let all joins be Sort-Merge or all be Grace-Hash. Then $MSO \leq q^4$ where q is the maximum Q-error taken over all intermediate results.

This guarantee provided the first quantitative bound wrt the ideal. However, an acute limitation is that the high-degree *quartic* dependency makes the guarantee impractically large when the Q-error is significant, as is often the case. Moreover, it is often not possible to apriori know the error value, making it infeasible to provide a bound to the user at query submission time. Second, the stringent requirement on the types of joins – all only Sort-Merge, or only Hash-Join – is an unrealistic assumption since query execution plans typically feature a mix of these joins, as well as others such as Nested Loops and Index Nested Loops. Finally, additional problematic issues related to how well Q-error minimization translates to actual query performance improvement, are detailed in Han *et al.* (2021).

5.2 Plan Bouquet

A radically different approach to bounding MSO was taken in the `PlanBouquet` algorithm (Dutt and Haritsa, 2016). The core idea here, similar to Smooth Scan, is to completely abandon the brittle selectivity estimation process. Instead, to carry out a runtime discovery of the selectivities, using a compile-time selected bouquet of plans. This technique lends itself to provable MSO guarantees even in situations where state-of-the-art systems may suffer arbitrarily poor execution. And what is even more attractive is that the guarantees are conservative, meaning that the empirical performance is well within these bounds even on industrial-strength environments. Similar to `CostGreedy`, `PlanBouquet` is also predicated on the PCM assumption.

We introduce the new approach through the example query of Section 2, augmented with a COURSE.`Fees` $<$ \$1 selection predicate, as shown in Figure 5.1. Here, \$1 is a parameter and modulating its value controls the selectivity of the COURSE table. Assume, for starters, that only the selectivity of this filter predicate is error-prone, whereas the join predicates are estimated correctly.

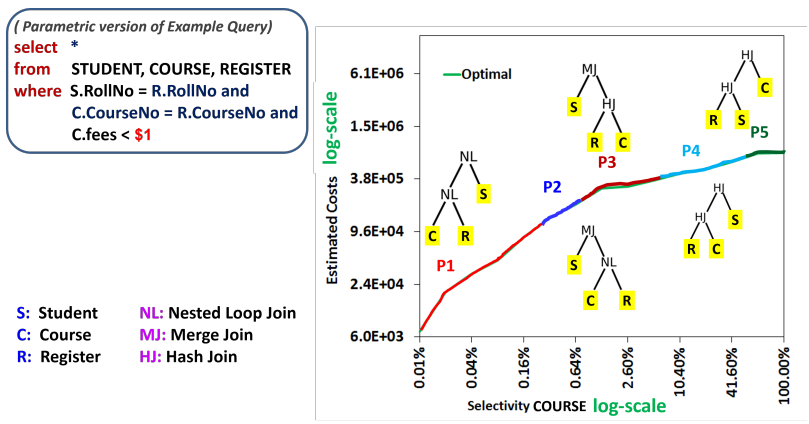


Figure 5.1: POSP Profile on 1D Selectivity Space (Dutt and Haritsa, 2016).

By assigning different values to the `Fees` parameter, and through repeated invocations of the optimizer, the POSP plans that cover the entire selectivity range of the COURSE table are identified. A sample

outcome of this process is shown in Figure 5.1, where the X-axis is the selectivity of COURSE (on a log scale) and the Y-axis is the optimal cost, as determined by the query optimizer, for executing the query (also on a log scale). Here, we find that the POSP set comprises plans P1 through P5, with each plan being optimal over a certain disjoint range. For instance, plan P3 is optimal in the (1.0%, 7.5%) selectivity interval. The join orders and implementations for each of these plans are also shown in the picture. Specifically, P1, which is close to the origin, computes a nested-loop join between COURSE and REGISTER, followed by another nested-loop join with the STUDENT table. This choice of plan is to be expected because it corresponds to the low selectivity region where the amount of data is small. As we go rightwards in the picture, the amount of data increases until a stage is reached where the optimizer recommends a switch to plan P2. In this plan, the second join with STUDENT becomes a merge-join and the left-deep tree structure morphs to a right-deep structure. And so on, until plan P5, where both the joins are hash-joins (again, this is to be expected due to the large amount of data at this high selectivity range). Further, the join sequence itself has changed to $((R \bowtie S) \bowtie C)$ from the $((C \bowtie R) \bowtie S)$ of plan P1.

5.2.1 Robustness Profile of Native Optimizer

When the performance of each of the region-specific optimal plans in the above example is extended over the entire selectivity space, the picture shown in Figure 5.2 is obtained.

Now, let us consider the worst-case behavior that could occur for the native optimizer over the selectivity space, and compare it with the ideal behavior. This comparison is reflected in the green (ideal) and red (worst) lines shown in Figure 5.3. Given the log-scale, the gap between these two lines is clearly quite large.

To explain how the red line was computed, for each point in the selectivity space, the POSP plan that is expected to suffer the *slowest* response time is identified. So, for instance, say the actual selectivity was 99% at runtime, but it was grossly mis-estimated to be 1%, then P1 would be used instead of the optimal P5, and the performance of

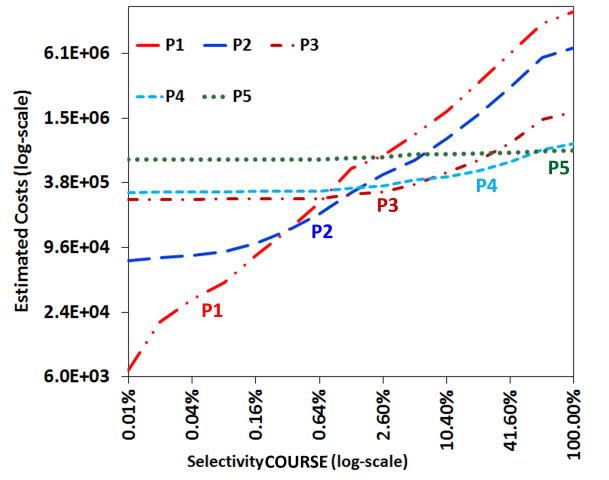


Figure 5.2: Extended POSP Cost Profile (Dutt and Haritsa, 2016).

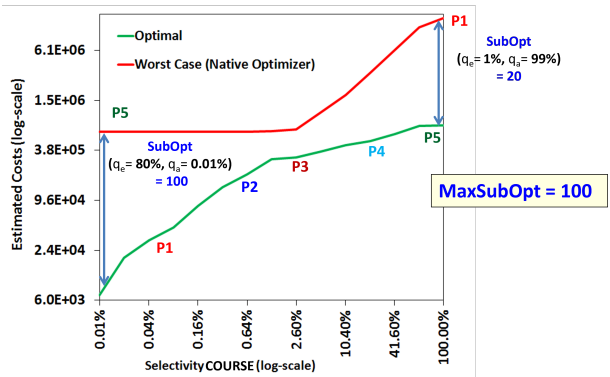


Figure 5.3: Sub-optimality Profile of Native Optimizer (Dutt and Haritsa, 2016).

P1 is 20 times worse than that of P5. On the flip side, if the actual selectivity was only 0.01%, but hugely over-estimated to be 80%, then P5 would be used instead of the optimal P1. Here, the performance of P5 is 100 times worse than P1. The red line shows such computations over the full selectivity space, and the maximum sub-optimality (MSO) is 100, near the origin.

5.2.2 Bouquet Identification

We now turn our attention to how `PlanBouquet` operates over the same selectivity space – this is visually shown in Figure 5.4. Given the ideal POSP cost profile from Figure 5.1, a series of horizontal lines are drawn parallel to the X-axis – essentially, the profile is *discretized*. Each of these lines is called an iso-cost (IC) line since it corresponds to a fixed cost. The lines are drawn such that each line has *double* the cost of the previous line and span the spectrum from the lowest cost (close to the origin) to the highest cost (at the maximum selectivity) – from this exercise, the iso-cost lines IC1 through IC7 are obtained. The points at which the iso-cost lines cut the POSP profile are marked by black squares, and the optimal plan at each of these locations is identified. As shown in Figure 5.4, the plans are P1, P2, P3 and P5, and they form the "plan bouquet". Note that plan P4 from the POSP profile is missing because its optimality interval falls *between* IC6 and IC7.

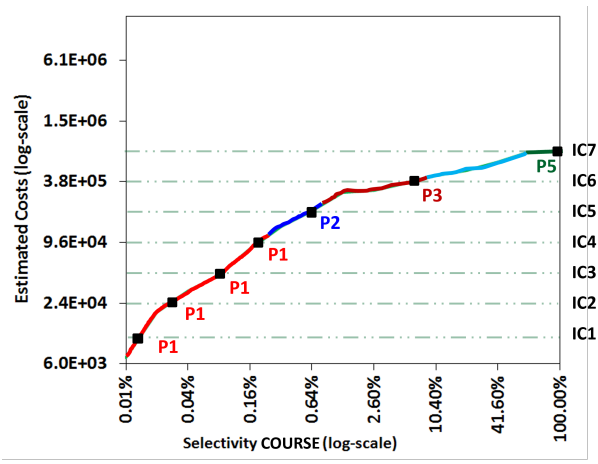


Figure 5.4: Bouquet Identification (Dutt and Haritsa, 2016).

5.2.3 Bouquet Execution

Given the above identification carried out at compile-time, the run-time execution of `PlanBouquet` operates in the following manner. Beginning

with the cheapest iso-cost line (i.e. IC1), the bouquet plan assigned to each intersection point is successively executed until either:

1. The partial execution overheads exceed the associated line's cost value – in this case, due to PCM, we know that the actual selectivity location lies beyond the intersection selectivity, motivating a switch to the next intersection point in the sequence; or

2. The current plan completes execution within the budget – in this case, we know that the actual selectivity has been reached, and that a plan which is at least *2-optimal* wrt the ideal choice was used for the final execution.

To make this process concrete, consider the case where the actual selectivity of COURSE.`Fees` is 5%. The algorithm begins by executing plan P1 until the execution overheads reach IC1 (1.2E4 | 0.015%). By virtue of PCM, the query will not complete within this allocated budget. Therefore, the execution is prematurely terminated, and the partial results (if any) produced thus far are *thrown away*. Then, the cost horizon is extended to IC2, which represents a doubling of the budget, and P1 is again executed until the overheads reach IC2 (2.4E4 | 0.03%). This process goes on until the overheads reach IC4 (9.6E4 | 0.2%). At this juncture, there is a change of plan to P2 as we look ahead to IC5 (1.9E5 | 0.65%). The new plan P2 is executed until the associated overhead limit (1.9E5) is reached. The cost horizon is now extended to IC6 (3.8E5 | 6.5%), in the process discarding P2's intermediate results and executing P3 instead. The execution with P3 completes before the cost limit is reached since the actual location, 5%, is less than the selectivity limit of IC6, namely 6.5%.

The total investment made by `PlanBouquet` in executing the query is 7.1 E5, computed as (0.12 E5+0.24E5 + 0.48 E5 + 0.96 E5 + 1.92 E5 + 3.4 E5). Whereas, the ideal algorithm would have only spent 3.4 E5. Therefore, viewed in toto, the net sub-optimality turns out to be $7.1/3.4 = 2.1$ since the exploratory overheads are 1.1 times the optimal cost, and the optimal plan itself was (coincidentally) employed for the final execution. Of course, with some obvious optimizations – for instance, the first four executions of P1 could be clubbed together with

a combined budget of 0.96E5, instead of going through the intermediate steps – the sub-optimality could be brought down to $6.3/3.4 = 1.8$.

A legitimate concern here could be that although the sub-optimality was only around 2 in the above example, there may be other selectivity locations where the sub-optimality is significantly worse. To investigate this concern, the complete sub-optimality of the `PlanBouquet` algorithm over the full selectivity space is shown in Figure 5.5. We see here that the blue line, which corresponds to `PlanBouquet` has a maximum sub-optimality of only 3.1 over the entire length, occurring around the 1% selectivity mark. Further, while an MSO of 3 may appear significant in isolation, it is much smaller compared to the 100 incurred by the native optimizer.

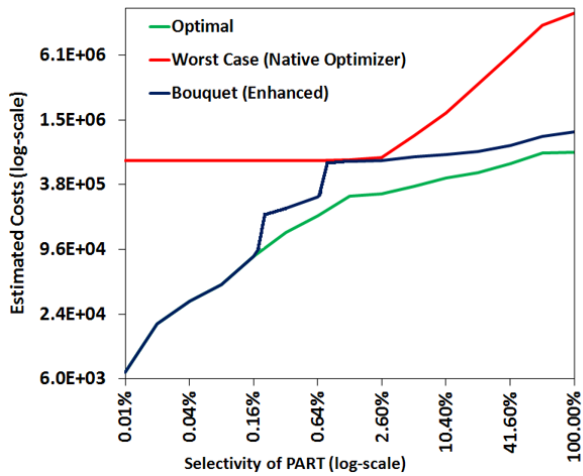


Figure 5.5: Sub-optimality Profile of `PlanBouquet` (Dutt and Haritsa, 2016).

5.2.4 MSO Guarantee

The good across-the-board performance of `PlanBouquet` has been established above for a particular query. However, it still begs the question "Is the performance of `PlanBouquet` much worse than the native optimizer for some *other* query?". To address this issue, a theoretical analysis of the `PlanBouquet` algorithm was presented in Dutt and Haritsa (2016), which we summarize here. Consider the picture in Figure 5.6, where a

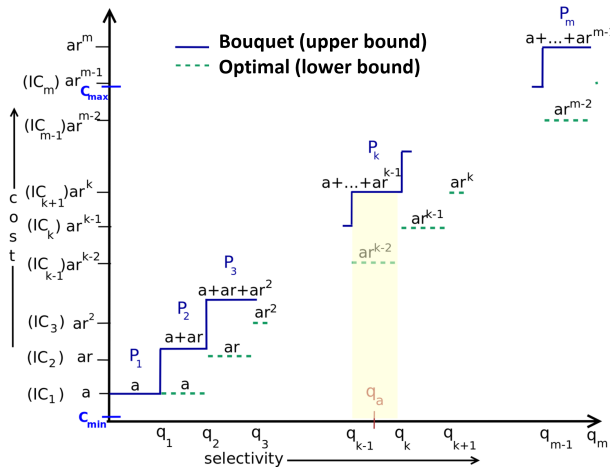


Figure 5.6: Worst-Case Analysis of `PlanBouquet` (Dutt and Haritsa, 2016).

generalized version of `PlanBouquet` is depicted – specifically, instead of a fixed cost-doubling regime, a generic geometric progression with initial value a and common ratio r is modeled:

We first mark on the X-axis, the selectivity locations q_i, corresponding to each of the iso-cost line intersections. Now assume wlog that the query has an actual (but unknown) selectivity q_a which lies in the range $(q_{k-1}, q_k]$. Given this formulation, the cost incurred by the ideal algorithm will be between IC_{k-1} and IC_k since these iso-cost lines have been drawn on the ideal POSP profile. The lowest possible value is IC_{k-1}, which given the geometric progression, corresponds to $a(r^{k-2})$.

Now let us consider the `PlanBouquet` performance. Here, the cumulative cost would be the costs incurred in moving up the "staircase" from IC_1 to IC_k. So, the worst-case cost would be $a + ar + ar^2 + ... + ar^{k-1}$, which sums to $\frac{a(r^k-1)}{r-1}$. Therefore, the sub-optimality is upper-bounded by

$$SubOpt(*, q_a) \leq \frac{\frac{a(r^k-1)}{r-1}}{ar^{k-2}} = \frac{r^2}{r-1} - \frac{r^{2-k}}{r-1} \leq \frac{r^2}{r-1} \tag{5.2}$$

Note that the dependence on k is removed in this formulation! It is now a simple matter to show that the RHS of Equation 5.2 reaches its minimum at $r = 2$, producing an MSO bound of 4.

Therefore, for *any* 1D query, `PlanBouquet` can guarantee an MSO bound of 4, and we emphasize again that this bound is wrt the optimal. Further, although the native optimizer may perform better than `PlanBouquet` on specific queries, it cannot provide any such inherent bounds on its worst-case performance.

At this point, another natural question is "Does there exist an alternative algorithm that can provide a tighter bound than `PlanBouquet`" – specifically, less than 4 for the 1D scenario. The answer is No since it can be proved that this bound is the best performance achievable by *any* deterministic online algorithm (Dutt and Haritsa, 2016). However, with the use of randomized algorithms, better probabilistic bounds, slightly less than 3, are achievable – see Dutt and Haritsa (2016) for details.

5.3 Multi-dimensional `PlanBouquet`

So far, our design and analysis of `PlanBouquet` was restricted to a single dimension. In moving to higher dimensions, both the design and analysis become more complicated. We discuss the case of 2D spaces here – the extension to higher dimensions is straightforward.

A sample of a 2D POSP profile is shown in Figure 5.7. We see here that the POSP profile has gone from being a line to a surface. And the iso-cost lines have now become horizontal planes that are parallel to the X-Y axial plane. What is particularly noteworthy is that the intersections of the iso-cost planes with the POSP surface may now have *multiple* plans appearing on the intersection. As an example, the intersection with one of the planes is shown with dotted lines in Figure 5.7, and there are 3 plans – colored blue, purple, and maroon – on the intersection contour.

A "top-down" view of the intersections leads to the generic picture shown in Figure 5.8 (a). Here, the iso-cost surfaces are represented by contours that represent a continuous sequence of selectivity locations (in contrast to the single location in the 1D case). Further, multiple bouquet plans may be present on each individual contour – as an example, on IC_k, there are four plans, P_1^k, P_2^k, P_3^k, P_4^k which are the optimizer's choices over disjoint (x, y) selectivity ranges on this contour. Here, if

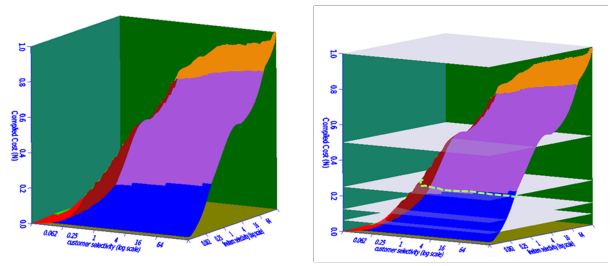


Figure 5.7: 2D Construction of `PlanBouquet` (Dutt and Haritsa, 2016).

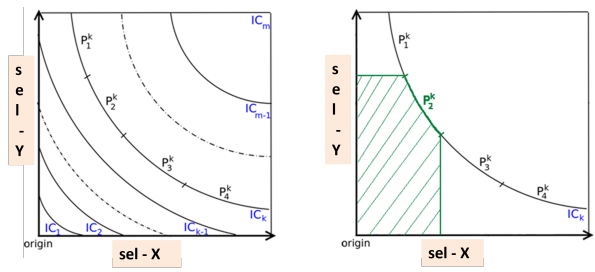


Figure 5.8: 2D selectivity space: (a) isocost contours, (b) plan prune space (Dutt and Haritsa, 2016).

P_2^k is executed with budget IC_k and the query does not complete, then it provably does not lie in the green hashed area shown in Figure 5.8 (b). However, we cannot jump to the conclusion that it lies beyond the IC_k contour since it may lie in the remaining *unshaded* regions below IC_k.

Therefore, to decide whether q_a lies below or beyond IC_k, *every* plan on the IC_k contour has to be executed – only if none complete, then the actual location definitely lies beyond the contour. The need for exhaustive execution is highlighted in Figure 5.9, where, for the four plans lying on IC_k, the regions in the selectivity space on which each of these plans is guaranteed to complete within the budget cost (IC_k) are enumerated (the contour superscripts are omitted in the figure for visual clarity). Note that while several regions are "covered" by multiple plans, each plan also has a region that it alone covers, denoted by the hashed regions in Figure 5.9. For queries located in such regions, only

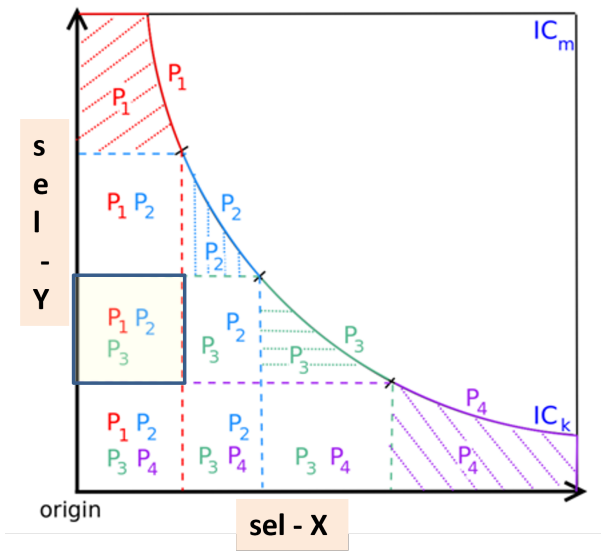


Figure 5.9: Prune space coverage by IC_k plans (Dutt and Haritsa, 2016).

the execution of the associated unique plan would confirm that the query is within the contour.

Performance Bounds Given a query Q with q_a located in the contour range $(IC_{k-1}, IC_k]$, the worst-case total execution cost for the multi-D bouquet algorithm is given by

$$C_{bouquet}(q_a) = \sum_{i=1}^{k} [n_i \times cost(IC_i)] \tag{5.3}$$

Using ρ to denote the number of plans on the *densest* contour, and upper-bounding the values of the n_i with ρ, the following performance guarantee is obtained:

$$C_{bouquet}(q_a) \leq \rho \times \sum_{i=1}^{k} cost(IC_i) \tag{5.4}$$

Now, following a similar derivation as for the 1D case, the following theorem results:

Theorem 5.2. Given a query Q with a multidimensional error-prone selectivity space, and the associated PIC discretized with a geometric progression having common ratio r and maximum contour plan density ρ, the bouquet execution algorithm ensures that: $MSO \leq \rho \frac{r^2}{r-1}$

Setting $r = 2$ in this expression ensures that $MSO \leq 4\,\rho$, providing a numerical compile-time guarantee.

5.3.1 Handling Large ρ

A problem in the above formulation is that the value of ρ could be quite large, typically in the hundreds, making the bound value impractically weak. An easy solution is to leverage the plan diagram reduction techniques described earlier in Section 4. Note that such reductions may mildly (within the λ factor) increase the average-case sub-optimality, but in return, substantially improve the worst-case behavior.

The MSO values, post-reduction, for a variety of TPC-H and TPC-DS-based query templates (listed in Dutt and Haritsa, 2018), with query dimensionalities ranging from 3 to 5, are shown in Figure 5.10. We see here that they are in the range of 10 to 50. While, at first glance, this may again appear large, note that these values are much lower than the sub-optimalities seen in practice with the native optimizers. To characterize it in lighter vein, an "absolutely terrible" situation has been improved to be "mildly horrible"!

5.3.2 Runtime Performance

While compile-time bounds were discussed thus far, we now consider how PlanBouquet actually behaves in run-time environments. In Figure 5.11, the MSO performance (on a log scale) of both the native PostgreSQL optimizer (maroon color) and PlanBouquet (green color) are shown for the query templates of Figure 5.10 on a PostgreSQL platform. The naming nomenclature for the queries is xD_y_Qz, where x specifies the number of dimensions, y the benchmark (H or DS), and z the query number in the benchmark. So, for example, 3D_H_Q5 indicates a three-dimensional error selectivity space on a query derived from Query 5 of the TPC-H benchmark.

	Query (dim)	MSO Bound
TPC-H	Q5 (3D)	14.4
	Q7 (3D)	14.4
	Q8 (4D)	33.6
	Q7 (5D)	43.2
TPC-DS	Q15 (3D)	14.4
	Q96 (3D)	14.4
	Q7 (4D)	19.2
	Q19 (5D)	38.4
	Q26 (4D)	24.0
	Q91 (4D)	43.2

Figure 5.10: MSO guarantees for benchmark-based templates (Dutt and Haritsa, 2016).

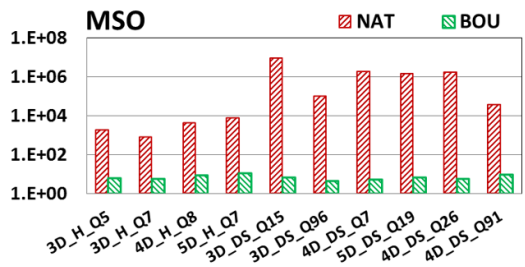


Figure 5.11: `PlanBouquet` performance on `PostgreSQL` (Dutt and Haritsa, 2016).

We observe orders-of-magnitude improvement by `PlanBouquet`– a particularly potent example is 3D_DS_Q15, where `PostgreSQL` has an MSO exceeding a million, whereas that of `PlanBouquet` is only around ten!

A similar experiment was conducted on a commercial database system. Since the engine's API does not directly support injection of selectivities, modified TPC-H queries 3D_H_Q5b and 4D_H_Q8b were constructed (Dutt and Haritsa, 2018), wherein all error dimensions correspond to selection predicates on the base relations.

The performance profile shown in Figure 5.12 was the outcome of this experiment. We see here that the MSO of the commercial system goes up to almost 10000, whereas `PlanBouquet` is in the low double digits.

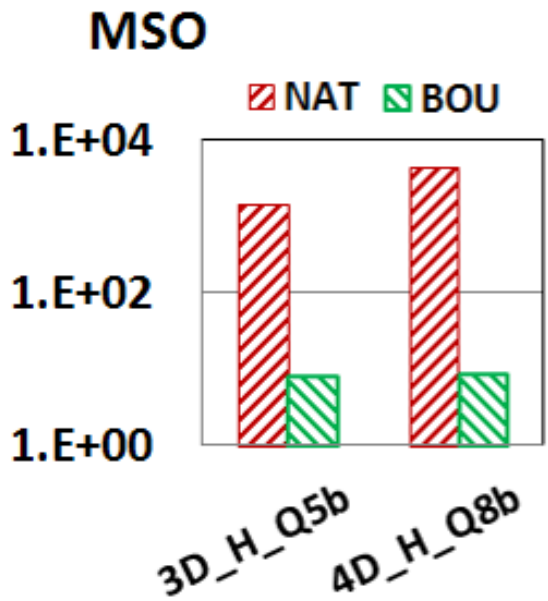


Figure 5.12: `PlanBouquet` performance on a commercial DBMS (Dutt and Haritsa, 2016).

5.4 Summary

Overall, `PlanBouquet` achieves, for the first time, bounded performance sub-optimality. Moreover, as highlighted previously, this is a numerical guarantee provided at compile-time. Second, it is inherently robust to changes in data distribution since selectivity estimations are completely eschewed. So, while the selectivity location may change based on distribution, the processing trajectory itself does not change since it is computed over the entire space. Third, it is amenable to non-invasive deployment, as a layer above the database engine. Finally, there

is repeatability in execution strategy, irrespective of the state of the metadata contents, a common source of regressions. This feature is particularly attractive in industrial settings, where stability, rather than ideal performance, is of paramount importance.

At first glance, the bouquet approach, with its partial execution of multiple plans, may appear similar to run-time re-optimization techniques such as POP (Markl *et al.*, 2004) and Rio (Babu *et al.*, 2005). However, a key difference is that they start with the optimizer's estimate as the initial seed, and then conduct a full-scale re-optimization if the estimate is found to be significantly in error. In contrast, `PlanBouquet` always starts from the origin of the selectivity space, and directly chooses plans from the bouquet for execution without reinvoking the optimizer.

The use of only one active plan (at a time) to process the data also makes `PlanBouquet` dissimilar from *routing-based approaches* – for example, plan-per-tuple (Avnur and Hellerstein, 2000) and plan-per-tuple-group (Polyzotis, 2005) – where data segments may be routed to alternative choices from a suite of concurrently active plans.

`PlanBouquet` may also superficially look similar to parametric query optimization (PQO) techniques, (e.g. PPQO Bizarro *et al.*, 2009), since a set of plans are identified before execution by exploring the selectivity space. The primary difference is that those techniques are useful for saving on optimization time for query instances with known parameters and selectivities. On the other hand, the goal of `PlanBouquet` is to regulate the worst-case performance impact when the computed selectivities are likely to be erroneous.

Finally, the bouquet technique does not modify plan structures at run-time. This is a major difference from the "plan-morphing" approaches, where the execution plan may be substantially modified at run-time using custom-designed operators, such as *chooseplan* (Cole and Graefe, 1994), *switch* (Babu *et al.*, 2005), and *feedback* (Chaudhuri *et al.*, 2008).

5.5 Limitations

While a bound of 4ρ has been achieved, there are some practical problems with regard to its identification and usage. First, ρ can be known only

by first constructing the plan diagram for the query template, which could be a very time-consuming affair, especially for higher-dimensional selectivity spaces. To some extent, this effort could be mitigated by not enumerating the whole space, but only the contours – an algorithm called Nexus for this purpose is provided in Dutt and Haritsa (2016). However, the inherent exponential nature of plan diagram enumeration remains. Second, the bound is not portable since the ρ value depends on the complexity of the specific plan diagram, which is a function of the underlying query optimizer, database and hardware system. These shortcomings are addressed in the `SpillBound` algorithm, described in the next section.

6

Structural Robustness Bounds

The `SpillBound` algorithm retains the core idea of `PlanBouquet`, namely, a cost-budgeted plan execution sequence guided by geometrically increasing iso-cost profiles over the optimal performance surface. However, unlike `PlanBouquet`, where the budgets were applied to plans in their entirety, the budget here is assigned specifically to maximize movement towards the (unknown) actual selectivity location *within* each plan execution.

The focused assignment is achieved, as shown in Figure 6.1, by leveraging a "spilling" mode of execution. That is, given an error-prone predicate whose selectivity is to be determined, the output of the corresponding operator is discarded and *not forwarded* to the downstream nodes of the execution plan tree. This means that the entire execution budget is utilized towards selectivity discovery since nothing is frittered away on processing of downstream nodes. Further, the spilling on the predicate is continued across successive contours until its selectivity value is fully known. That is, selectivities are sequentially identified over the plan tree.

An immediate question that arises is the sequence in which these error-prone predicates should be chosen for spilling. A bottom-up exe-

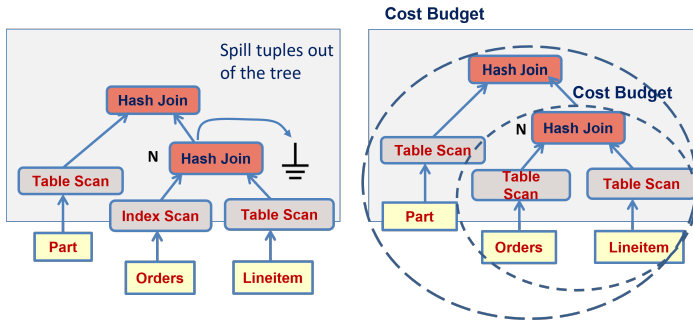


Figure 6.1: Spilling Mode of Execution (Karthik *et al.*, 2019).

cution pattern is followed, whereby the lowest predicate in the tree is chosen first, and then we progressively traverse up the tree. Specifically, a total ordering of the predicates is obtained by first partitioning the plan into a sequence of pipelines, and then ordering the predicates within each pipeline. An example based on TPC-DS Q26 is shown in Figure 6.2, where there are four pipelines, L_1 through L_4, highlighted by dotted ovals, executed in this order. The nodes are represented as N_i and in this figure the total ordering is N_{10}, N_9, N_4, N_3, and their selectivities are learned one by one.

This iterative process ensures that whenever a node is chosen for spilling, the selectivities of all the nodes below it are known correctly, either because (a) they are not error-prone, or (b) they have been previously learned fully in the ongoing discovery process.

6.1 Half-space Pruning

With the above spilling mode of execution, a much stronger *half-space pruning* is achieved in the selectivity space, as opposed to the *hypo-graph* pruning delivered by `PlanBouquet` – this improvement is visually shown in Figure 6.3. The extent of selectivity movement depends on the plans chosen from the contour. To maximize movement, for each dimension, the contour plan that spills the farthest in that dimension is chosen. It can be shown that with this approach, crossing a *D-dimensional* contour requires at most D plan executions. This is in marked contrast to `PlanBouquet` where *all* plans on the contour had to be executed before a crossing could be made to the next contour.

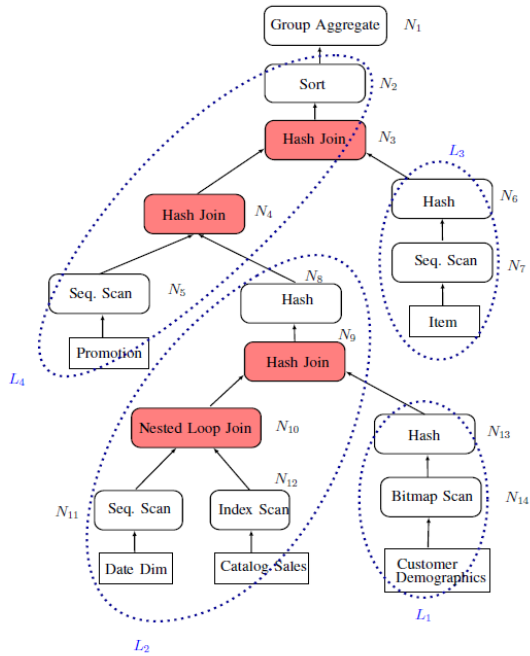


Figure 6.2: Spill Predicate Sequencing (Karthik *et al.*, 2019).

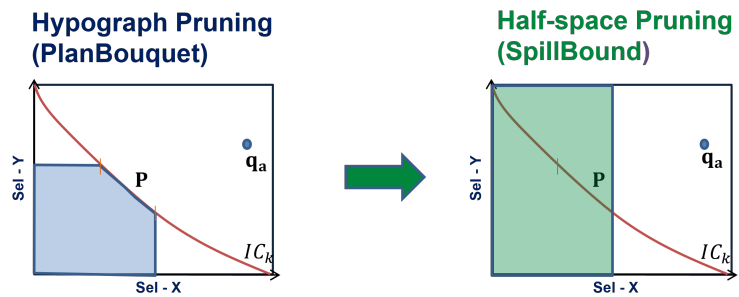


Figure 6.3: Hypograph to Half-Space Pruning (Karthik *et al.*, 2019).

Overall, thanks to the pair of complementary techniques – half-space pruning and limited contour executions – it can be proved (Karthik *et al.*, 2016) that:

$$MSO \leq D^2 + 3D \tag{6.1}$$

This means that the MSO now only depends on the *dimensionality* of the space, and not on its contents! That is, in addition to being data-independent and metadata-independent, which was inherited from `PlanBouquet`, *platform-independence* has been achieved as well (under the assumption that D remains constant across the platforms). In a nutshell, there has been a movement *from behavioral bounds to structural bounds*.

Although `SpillBound` appears to provide stronger robustness guarantees than `PlanBouquet` in theory, the question could still be asked whether, in practice, is it the other way around? This issue is addressed in Figure 6.4, where over a suite of TPC-DS-based queries, `SpillBound` performs substantially better on most of them. For instance, consider query 5D_Q29 – `PlanBouquet` has an MSO of 42, whereas `SpillBound` is only 15.

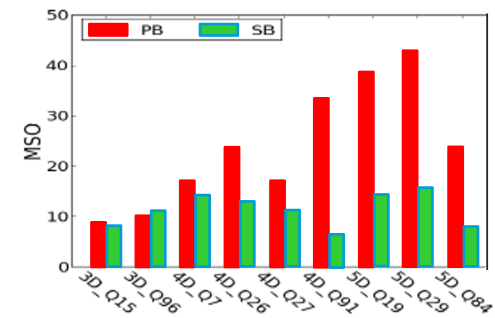


Figure 6.4: Empirical comparison of `PlanBouquet` and `SpillBound` (Karthik *et al.*, 2019).

Finally, a *lower-bound* that is linear in D for this half-space pruning class of algorithms is derived in Karthik *et al.* (2016). This shows that `SpillBound` is within an $O(D)$ factor of the ideal, extracting much of the power of the new approach.

6.2 Ad-hoc Queries

Both `PlanBouquet` and `SpillBound` are predicated on apriori possessing a plan-diagram, with enumerated iso-cost contours, to identify the plan

bouquet. Due to the significant computational cost of creating the diagrams, these algorithms become suitable only for "canned" queries that are invoked repeatedly. In order to provide robustness for "ad-hoc" queries as well, a new technique, called `FrugalSpillBound`, was presented in Venkatesh *et al.* (2018). Here, recourse is taken to a stronger geometric property of plan cost functions. Specifically, in addition to monotonicity, it is also assumed that PCFs exhibit *concave-down* behavior with monotonically non-decreasing slopes. That is, the cost of processing each additional tuple is less than that of the prior tuples. Through experimentation with benchmark environments hosted on contemporary database engines, it was established that such concavity is the norm.

With the above assumption, instead of computing the precise location of the intersections of the iso-cost surfaces with the optimal cost profile, an approximate proxy iso-cost surface called a *bounded contour-covering set* (BCS) is dynamically found. The BCS for a contour is defined as the set of locations such that:

1. Every location in the contour is *spatially dominated* by at least one location in this set; and
2. The cost of each location in BCS is *bounded* to within an η factor of the contour cost.

Defining the BCS of contour $\mathcal{IC}_i$ by BCS_i, the following condition needs to be satisfied by BCS_i:

$$\forall q \in \mathcal{IC}_i, \exists\ q' \in BCS_i \text{ such that } q \preceq q' \text{and } Cost(P_{q'}, q') \leq \eta \texttt{CC}_\texttt{i}$$

where $\texttt{CC}_\texttt{i}$ is the cost of isocost contour $\mathcal{IC}_i$. To make this notion concrete, a candidate BCS_i for the example contour $\mathcal{IC}_i$ shown in Figure 6.5, is $\{c_1, c_2, c_3\}$ which covers the entire contiguous length of the contour. As a specific case in point, the covering location c_2 fully covers the optimality segments of P_5 and P_6, as well as parts of P_4 and P_7, in $\mathcal{IC}_i$.

Leveraging the concavity property, it is feasible to efficiently identify the BCS for each contour. Further, it is shown in Venkatesh *et al.* (2018)

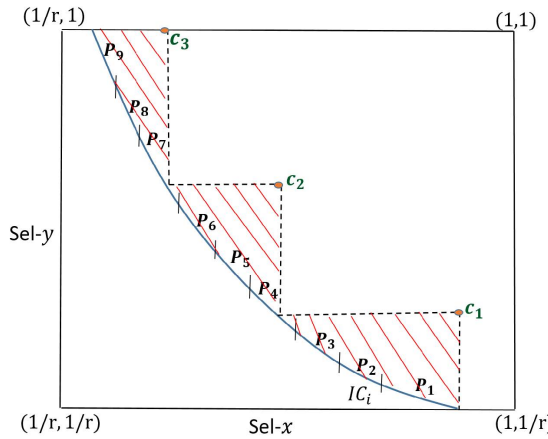


Figure 6.5: Bounded Contour-covering Set (BCS).

that for an increase by at most η in MSO guarantees, the overheads reduction factor is at least

$$\begin{aligned} \gamma &= r/\log_\eta r && D = 1 \\ \gamma &= \Omega(r^D/(D\log_\eta r)^{D-1}) && D \geq 2 \end{aligned} \tag{6.2}$$

That is, the initial regime of `FSB` provides an *exponential improvement* in γ for a *linear degradation* in η.

So, when $\eta = 2$, γ can provide two orders of magnitude reduction in overheads. Beyond this theoretical guarantee, γ was empirically found in Venkatesh *et al.* (2018) to often reach close to *four* orders of magnitude. As a practical matter, for $D = 5$ and $r = 100$, the compilation efforts reduced from a few days to a few minutes on contemporary servers.

6.3 Linear Guarantees

`SpillBound` provides a robustness guarantee that is quadratic in the selectivity space dimensionality, D, but the lower bound is linear in this parameter. Therefore, it is natural to ask whether there exist environments in which the lower bound can be achieved. To investigate this issue, a concept called *contour alignment* was introduced in Karthik *et al.* (2019). A contour is said to be aligned if the contour plan that

is incident on the boundary of the parameter space, has its selectivity learning dimension (during spill-mode execution) matching with the incident dimension. For instance, in Figure 6.5, if plan P_1 and P_9 provide selectivity movement along the X and Y dimensions, respectively.

Given the above characterization, the MSO bound can be tightened to linear if the contour alignment property is satisfied at every contour in the selectivity space. Specifically, the bound is now $\mathbf{2D + 2}$. Unfortunately, in practice, the alignment property may not be satisfied at all contours. Therefore, techniques to force alignment through explicit induction were also presented in Karthik *et al.* (2019) – however, such forcible alignment may sometimes entail a severe performance penalty, so it can be employed only in select cases where the alignment can be obtained cheaply.

6.4 Summary

This section showed that by (a) making minimally invasive changes to the execution engine, and (b) stronger, but practically realistic, geometric assumptions about plan cost behavior, it is feasible to upfront provide strong platform-independent performance guarantees for both canned queries and ad-hoc queries.

7

Robust Cost Models

Thus far, our focus has been on incorporating robustness with respect to the *operator cardinality model*, which is primarily responsible for the poor choice of runtime plans. However, robustness can also be adversely affected by errors in the *operator cost model*, and this issue has been the subject of several studies during the past decade. As mentioned previously, the two models address very different aspects of the data processing environment – the cardinality model reflects the ability to capture the distributions and correlations present in the data, whereas the cost model registers the ability to capture the behavior of the underlying hardware and physical operator implementations.

There have been two schools of thought on addressing cost modeling errors – one advocating learning-based mechanisms (e.g. Ganapathi *et al.*, 2009; Akdere *et al.*, 2012; Sun and Li, 2019), and the other proposing that fine-tuning of the classical statistical approaches is sufficient to obtain viable models (e.g. Wu *et al.*, 2013b; Wu *et al.*, 2013a; Wu *et al.*, 2014).

The claim made by the learning group was that the current cost models, which are based on direct scaling between the optimizer estimated costs and the actual running times, are largely unusable. As empirical

evidence of this shortcoming, they presented sample evaluations, like the one shown in Figure 7.1. In this figure, obtained on `PostgreSQL`, the actual execution time is plotted against the optimizer's estimate of this time. It is clear that there is no simple linear fit that can be interpolated between the two quantities, and in fact, the average error is well in excess of 100 percent!

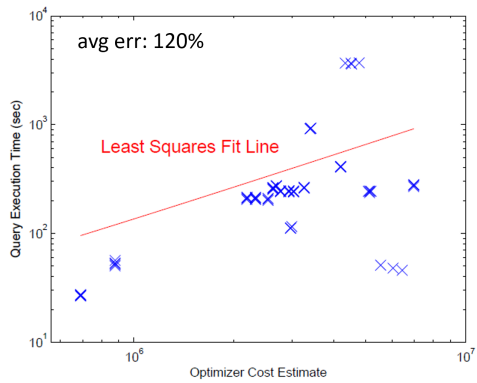


Figure 7.1: Poor Accuracy of Cost Model (Akdere *et al.*, 2012).

However, there was subsequently a pushback to this conclusion from the statistical community. To understand their argument, consider the following formula which is the cost model of `PostgreSQL`:

$$C = n_s c_s + n_r c_r + n_t c_t + n_i c_i + n_o c_o \tag{7.1}$$

where the c_x are the unit costs for various disk and CPU operations, as shown in Table 7.1. The corresponding n_x are the number of such operations. A normalized value of 1.0 is assigned to c_s (sequential page access), and the values of the remaining parameters are set relative to c_s.

The formulation in Equation 7.1 is usually converted to a wall-clock time prediction, T, in the following manner:

$$T = a.C = a(n_s + n_r \frac{c_r}{c_s} + n_t \frac{c_t}{c_s} + n_i \frac{c_i}{c_s} + n_o \frac{c_o}{c_s}) \tag{7.2}$$

where a is a proportionality constant. Note that this formulation allows for directly using the scaled values listed in Table 7.1.

Table 7.1: Cost Parameters.

Parameter	Cost Unit	Scaled Value
c_s	Sequential Page Access	1.0
c_r	Random Page Access	4.0
c_t	CPU tuple access cost	0.01
c_i	CPU index access cost	0.005
c_o	CPU operator cost	0.0025

Given this framework, it was mooted that the reason for direct scaling to fail was not the approach itself, but an artifact of (a) incorrect ratios between the c_x values, and (b) incorrect n_x values. The proposed solution to these problems was accurate *calibration* of these values.

Before we get into the mechanics of calibration, a sample evaluation of how the calibrated version works against the scaled version is shown in Figure 7.2. As is evident, unlike the scaled version, the calibrated version has a strong linear relationship between the actual execution time and the predicted time. A related benefit of such calibration is that even if a sub-optimal plan is chosen due to incorrect cardinality estimates, we can at least accurately estimate the running time of this chosen plan.

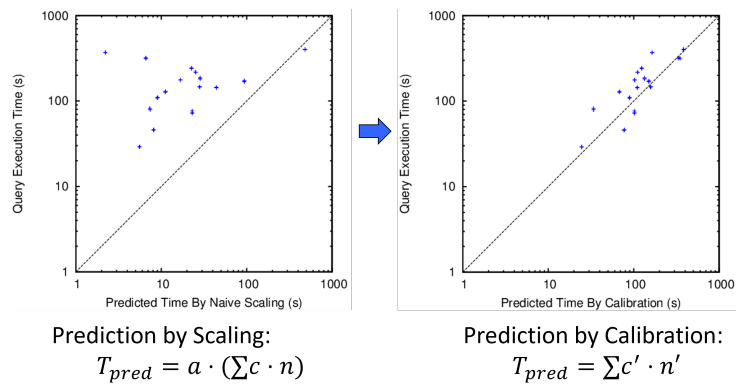


Figure 7.2: Prediction Quality: Scaling vs Calibration (Wu, 2013).

In order to achieve calibration, we need the correct c_x values and this can be done in an offline fashion through the use of appropriate

profiling queries. And the accurate n_x values can be computed after the plan has been decided through an online *sampling* exercise. This system architecture is captured in Figure 7.3.

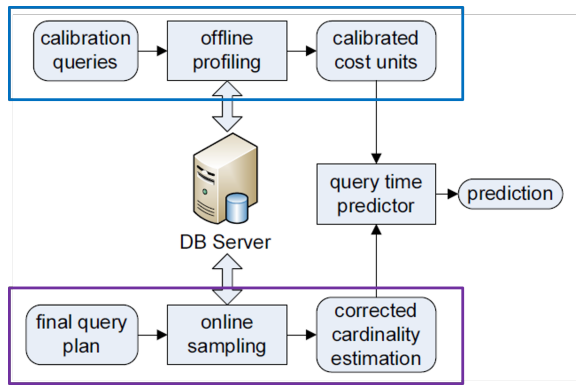


Figure 7.3: Cost Model Calibration Architecture (Wu *et al.*, 2013b).

7.1 Calibrating Unit Cost Parameters

Obtaining the calibration for the c_x values is surprisingly simple. The five profiling queries on a generic table R shown in Figure 7.4, when executed in the given sequence, are sufficient to provide these values. The basic idea here is to isolate the various parameters and progressively solve for them, one by one. The queries are chosen to achieve *completeness* – each c_x should be covered by at least one query; *conciseness* – set of queries is incomplete if any query is removed; and most importantly, *simplicity* – each query should be as simple as possible, both for efficiency and for unambiguous measurement.

For instance, the first query captures an in-memory sequential scan, and therefore establishes the value of c_t because both the running time and the number of tuples (n_{t1}) can be accurately measured (or, even simpler, n_{t1} is directly available from the metadata catalogs). Then, armed with this knowledge, we can now execute the second query, where in addition to the scan, there is an aggregation operator – this leads us to the c_o value since we know that n_{o2} and n_{t2} are both equal to the known row-cardinality of R. And so on, until we get all five parameters.

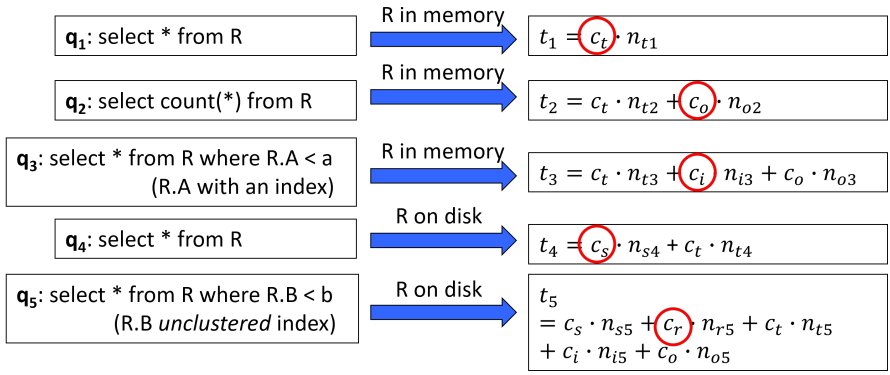


Figure 7.4: Calibration Queries for Unit Cost Parameters (Wu *et al.*, 2013b).

7.2 Calibrating Number of Operations

At each operator, the number of operations executed is a function of its input cardinalities. For instance, as shown in Figure 7.5, the number for in-memory Sort is $n_o = 2 * N_t * log(N_t)$, where N_t is the input cardinality to the operator. Similarly, in Nested Loops join, the number of tuples fetched is proportional to the product of the row-cardinalities of the input relations.

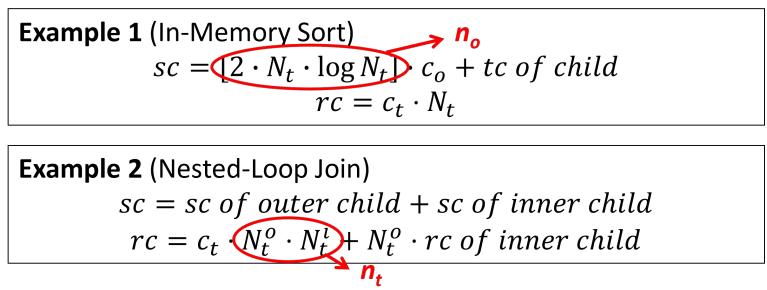


Figure 7.5: Cardinality Calibration for Operators (Wu *et al.*, 2013b).

Now, calibrating the input cardinalities brings us back to the original problem of cardinality estimation. And one could take the position

that this problem should be addressed first. However, that would require accurate estimations for *all* the plans in the exploration space, a daunting task. Instead, we now try to produce accurate estimations for the (potentially sub-optimal) chosen plan – since only one plan has to be worked with, we can afford to spend extra time on getting it right. In particular, instead of the coarse histograms typically used in cardinality models, the more heavy-duty approach of sampling can now be employed. A subtle but important point to note here is that modest cardinality estimation errors are tolerable in plan selection since our interest is only in the *relative ranking* among plans, whereas precise values are required for cost estimation since an *absolute* time value is being predicted.

7.2.1 Sampling Estimator

The sampling estimator for a join operator on tables R_1 and R_2 is shown in Figure 7.6, leveraging a classical result from Haas *et al.* (1996). Here, the tables R_1 and R_2 are first broken up into block-based partitions. Then, a pair of random partitions from the two tables is picked – in the picture, these first picks are B_{11} and B_{22}. The chosen blocks are brought into memory and joined. The selectivity for this pair is computed as the number of joined rows normalized to the product of the partition sizes. This sampling is carried out repeatedly – in the figure, there are k such pairs – and their selectivities are computed. Finally, the average of the partition selectivities is taken as the final estimated selectivity.

It can be shown that the proposed estimator is not only unbiased, but more importantly, is strongly consistent, meaning that the estimate only gets better (closer to the true value) with each additional sample.

Refinements

While the above estimator works well wrt quality, there are some practical problems in its usage. First, accessing the chosen partitions at runtime involves random IOs and this could be time-consuming. An easy solution is to take the samples offline and store them as new tables in the database. Second, query plans contain a tree of operators, and carrying out the estimation process for each operator from scratch

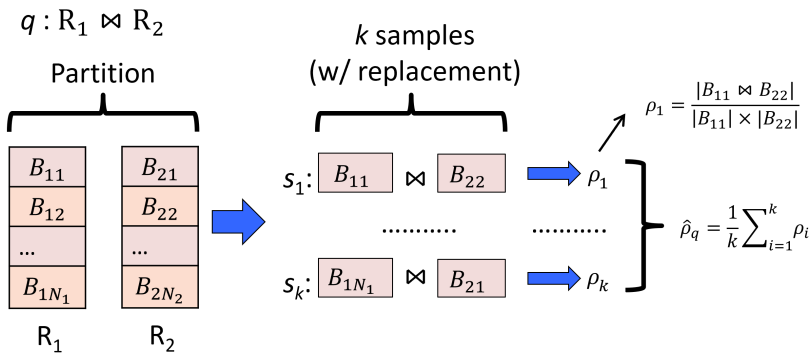


Figure 7.6: Sampling-based Cardinality Estimator (Wu *et al.*, 2013b).

could again be time-consuming. This issue can be solved by estimating multiple operators in a single run by reusing the partial results obtained from the previous operators. Finally, the estimator is applicable for selects and joins only, which means that operators such as aggregates and projections are outside the ambit. However, the good news is that operators of this type usually appear at the top of the tree, and therefore if the existing models of the query optimizer are used for their estimation, the errors are typically limited.

An example of this refined estimator is shown in Figure 7.7. Here, the original query plan chosen by the optimizer is shown on the left, computing a left-deep join of R_1, R_2 and R_3 in sequence followed by an aggregation. This plan is rewritten with the sampled versions of the three tables – R_1^s, R_2^s, R_3^s – and then executed. The selectivities for the first and second joins are computed as per the estimator from Haas *et al.* (1996). However, note that the output size of the first join is reused in computing the second join's selectivity. Essentially, the selectivity computations are pipelined through the plan tree, and whenever an operator that is out-of-scope appears – such as the aggregation in this query – the standard models from the optimizer are used. An important point to note is that these models now operate on the refined input estimates from q_2, and therefore the aggregation estimator is also positively impacted by the refinements of the join estimates.

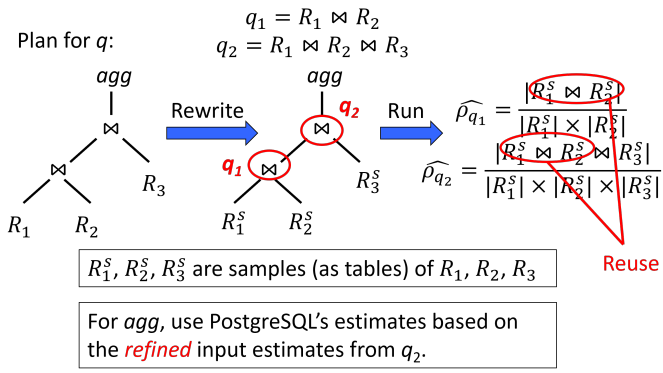


Figure 7.7: Refinement of Cardinality Estimation (Wu *et al.*, 2013b).

7.3 Performance

To demonstrate that accurate calibration is required prior to use of cost models, the cost parameters for `PostgreSQL` were evaluated in Wu *et al.* (2013b) for two hardware platforms, PC1: Single-core 2.27 GHz Intel CPU with 2 GB memory, and PC2: 8-core 2.40 GHz Intel CPU with 16 GB memory. The results are shown in Figure 7.8, and compared with the default values used by the optimizer. As is evident, there are substantive differences in these values, even extending to order-of-magnitude changes – for instance, the c_i in PP1 is 6E-3 as compared to the default value of 5E-2! Further, there are also large differences between the two platforms – for instance, c_r is 1.2 in PC1 and 9.7 in PC2.

We now look at the performance of this calibrated version over entire queries. The precision metric used is the *Mean Relative Error*:

$$MRE = \frac{1}{M} \Sigma_{i=1}^{M} \frac{|T_i^{pred} - T_i^{act}|}{T_i^{act}} \tag{7.3}$$

Although this metric was used in previous studies as well, a question that could be raised is about its suitability. Consider a trivial estimation algorithm that always gives the predicted value as 0, *irrespective* of the query. Despite this ridiculous approach, note that the MRE will always be *one*. A better error metric would have been, similar to cardinality estimation, the Q-error metric.

PC1:

Cost Unit	Calibrated (ms)	Calibrated (normalized to c_s)	Default
c_s: seq_page_cost	5.53e-2	1.0	1.0
c_r: rand_page_cost	6.50e-2	1.2	4.0
c_t: cpu_tuple_cost	1.67e-4	0.003	0.01
c_i: cpu_index_tuple_cost	3.41e-5	0.0006	0.005
c_o: cpu_operator_cost	1.12e-4	0.002	0.0025

PC2:

Cost Unit	Calibrated (ms)	Calibrated (normalized to c_s)	Default
c_s: seq_page_cost	5.03e-2	1.0	1.0
c_r: rand_page_cost	4.89e-1	9.7	4.0
c_t: cpu_tuple_cost	1.41e-4	0.0028	0.01
c_i: cpu_index_tuple_cost	3.34e-5	0.00066	0.005
c_o: cpu_operator_cost	7.10e-5	0.0014	0.0025

Figure 7.8: Calibrated Unit Costs on Platforms PC1 and PC2 (Wu *et al.*, 2013b).

Modulo the above criticism, there are three variants of the calibrated estimator: First, an idealized E_t where the c_x parameters are calibrated, and the n_x values are the true numbers (obtained through a prior query execution). Then, E_o, where the c_x parameters are calibrated and the n_x values are obtained from the optimizer. And finally, E_s^f, where the c_x parameters are calibrated and the n_x values are obtained by the sampling technique, with f being the sampling ratio at each of the base tables.

The above variants were evaluated against both scaling-based and learning-based approaches, and a sample performance is shown in Figure 7.9 on the baseline TPC-H database, which has uniform and uncorrelated data distributions. Here, E_{SVM} and E_{REP} are estimators based on the learning techniques of Support Vector Machines and REP Trees, while E_o^{LR} is based on simple linear regression corresponding to the direct scaling approach.

In this figure, we first observe that E_o, the native optimizer's cost model post-calibration, itself does very well, close to the ideal E_t. This is not surprising given that the optimizer's assumptions of uniformity and independence hold on the TPC-H database. Accordingly, the incorporation of sampling, represented by $E_s^{0.1}$ and $E_s^{0.3}$, does not really help in this case. However, what is really surprising is that the learning-based techniques do very poorly for some of the queries, as indicated by the large error bars. In fact, their MRE values over the workload almost

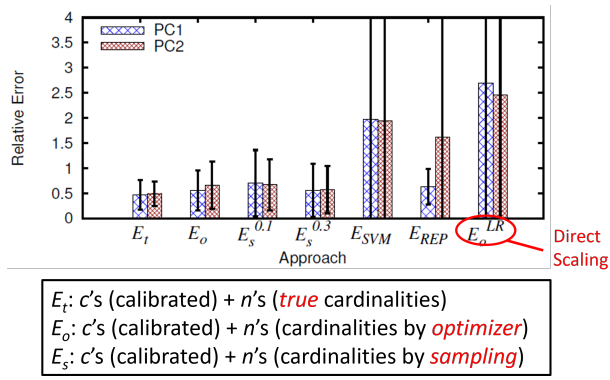


Figure 7.9: Accuracy on Uniform Data (Wu *et al.*, 2013b).

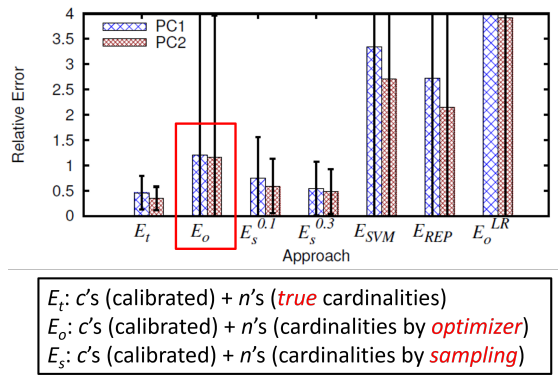


Figure 7.10: Accuracy on Skewed Data (Wu *et al.*, 2013b).

reach 2. Now recall the trivial algorithm that we had previously posited, which would always give a selectivity prediction of 0 – that algorithm would have fared better than the sophisticated learning approaches since its MRE is capped at 1! At a meta-level, such results point out that machine learning is not a panacea for database modeling issues, and can often turn out to be brittle in the extreme. Finally, considering the scaling approach E_o^{LR}, we find its performance is even worse, with the MRE close to 3.

When the same experiment is carried out with skewed data, the performance profile shown in Figure 7.10 is obtained. We now see E_o performing poorly compared to E_t. Again, this should be expected since

the optimizer's distributional assumptions are significantly violated. However, when sampling is used to reduce the cardinality estimation errors, as reflected in $E_s^{0.1}$ and $E_s^{0.3}$, the prediction accuracy becomes almost as good as E_t. Turning to the learning algorithms, E_{SVM} and E_{REP}, we find their performance is worse than even the uniform data scenario, with the MREs reaching 3 and beyond. Finally, the direct scaling approach, E_o^{LR}, continues to be the worst with an MRE of around 4.

In summary, the above results illustrate that using careful calibration in conjunction with statistical models offers visibly superior performance to learning approaches. Moreover, statistical models are also preferable from an efficiency perspective because of their negligible computational overheads at run-time.

8

Machine Learning-based Techniques

During the past few years, driven by the amazing success of machine learning in a variety of domains, a flood of papers advocating *machine learning-based* approaches to query processing has appeared in the research literature (e.g. Dutt *et al.*, 2019; Havenstein *et al.*, 2020; Hayek and Shmueli, 2020a; Kiefer *et al.*, 2017; Kipf *et al.*, 2019b; Marcus and Papaemmanouil, 2019; Negi *et al.*, 2020; Sun and Li, 2019; Yang *et al.*, 2019). The basic idea is to replace the classical coarse parametrized models with fine-grained learned models. The expectation is that these deep models are better able to capture the in situ data and system behavior due to their flexibility, scalability and lack of prior assumptions.

Within this corpus, there are two broad classes – the first is *query-based* techniques, a supervised form of learning. Here, the models are constructed by training on a large set of queries and then using the observed values for cardinality and/or cost during execution as the labels. An alternative approach is to use a purely *data-based* learning technique, which falls under the unsupervised category. Here, the objective is to model the joint probability density function of the underlying data so as to capture the distributions and correlations that are critical to coming up with precise estimates. Finally, there are also *hybrid models* that leverage both queries and data in their learning process.

In the remainder of this section, we cover a representative technique from the above classes – `MSCN` (Kipf *et al.*, 2019b) for query-based methods and `NARU` (Yang *et al.*, 2019) for data-based methods.

8.1 Query-based Models

In `MSCN` (Multiset Convolutional Neural Network), the emphasis is on modeling cardinalities for correlated joins since they are particularly difficult to model well and have a critical impact on the overall quality of the estimation process. For instance, the observation that *"French actors are more likely to feature in romantic movies than actors of other nationalities"* may be found in the IMDB movie database.

The foundation of the `MSCN` approach is to leverage the Deep Sets learning framework (Zaheer *et al.*, 2017), which is a neural network module for operating on sets. The framework is based on the notion of *set convolution*, wherein any permutation-invariant function $f(S)$ on a set S is decomposable into the form $\rho[\sum_{x \in S} \phi(x)]$, with ρ and ϕ being appropriately chosen functions. In the relational world, this means that both $(A \bowtie B) \bowtie C$ and $A \bowtie (B \bowtie C)$ are represented as $\{A, B, C\}$. The other key idea is to integrate sampling – specifically, the bitmaps of qualifying base table samples – in the feature set, during both training and testing.

The advantage of this approach is that the techniques work in a complementary manner to learn join-crossing correlations. Additionally, it addresses the "0-tuple" problem often encountered in sampling, where very few or even zero tuples qualify the selection predicate, risking large errors in estimation. In essence, the query features can be relied upon in such low-frequency scenarios.

8.1.1 Training Data

To obtain the training data, synthetic queries are created using schematic information such as data types and constraints, as well as actual data values from the application database. Then these queries are executed and the true cardinalities are obtained via these executions. Further, the queries are annotated with bitmaps indicating qualifying base table samples that contributed towards producing the final result.

8.1.2 Feature Selection and Representation

A query is represented as a collection of a set of tables, a set of joins and a set of predicates. All logical query features, including tables, filter predicates and joins, are one-hot encoded. Whereas all numerical values, including the query literals and the true cardinalities, are normalized to the $[0, 1]$ range.

An example query and its encoding are shown in Figure 8.1 on the IMDB database. Here, there is a join between TITLE and MOVIE_COMPANIES on `movie_id`, as well as filter predicates on `production_year` and `company_ id`. The table identifiers are represented using one-hot encoding. If we assume that there are three tables in this database, the encoding is 010 for TITLE and 001 for MOVIE_COMPANIES.

A bit vector is associated with each sampled base table, indicating which tuples contributed towards the query result. These bit vectors are shown adjacent to the `table_id` – for instance, the first tuple from TITLE and the last tuple from MOVIE_COMPANIES are both productive tuples, whereas the last tuple from TITLE and the first tuple of MOVIE_COMPANIES are sterile, as indicated by the 1 and 0 in those positions, respectively.

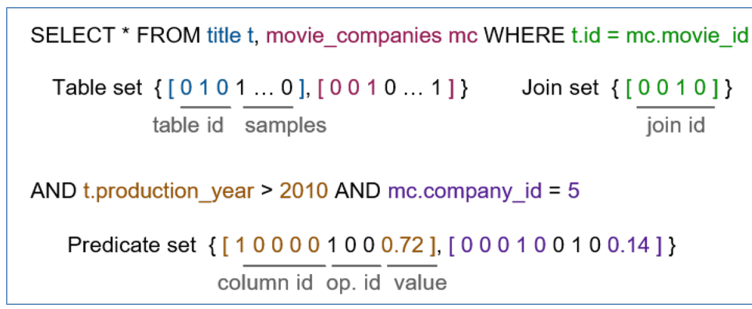


Figure 8.1: `MSCN` Query Encoding (Kipf *et al.*, 2019b).

Each edge of the schematic join graph is one-hot encoded, and the particular join instantiated in the query is encoded as 0010. For filter predicates, the representation is $column\_id, operator\_id, normalized\_value$ with the identifiers being one-hot encoded. Accordingly, the `production_year` and `company_id` predicates are assigned column iden-

tifiers 10000 and 00010, respectively (since there are five columns in both TITLE and MOVIE_COMPANIES). The operator = is encoded as 010 and > as 100. Finally, the value 2010 is normalized to 0.72 based on the minimum and maximum values in `production_year`, while the value 5 is normalized to 0.14 based on the corresponding values in `company_id`. Finally, the true result cardinality of 665 is normalized to 0.1 since the maximum feasible output size for this query is 6650.

8.1.3 Learning Model

The learning model of `MSCN` is shown in Figure 8.2. For each of the different sets in the model space – namely, *table* set, *join* set, and *predicate* set – a separate module is created, comprising a standard two-layer neural network per set element with shared parameters and ReLU activation functions. For each module, its individual element outputs are averaged to reduce the subsequent computational efforts and representational complexity. The set-specific averages are then concatenated and fed into a final output network. This network is also two-layer but has a final Sigmoid activation function to obtain the maximum variability for modeling non-linearity.

The optimization metric is Q-error, and the goal is to minimize its mean value on the training query set. This means that the robustness is in expectation over the workload, and not on individual queries.

8.1.4 Performance

The `MSCN` approach has been evaluated on the IMDB dataset which contains several correlations and is therefore a "tough-nut" for cardinality estimators. A sample performance profile is shown in Figure 8.3 for number of joins going from 0 to 2, along with equality and range predicates. The comparative baseline is the native `PostgreSQL` engine. On the Y-axis is the Q-error (on a log-scale) with overestimations appearing above the ideal value of 1, and underestimations below 1. This 0-to-2 join regime corresponds to the training data.

If we consider `PostgreSQL`, it performs quite well for zero or single join queries, although even in these very simple cases there are a few outliers. However, when the number of joins is increased to two, the

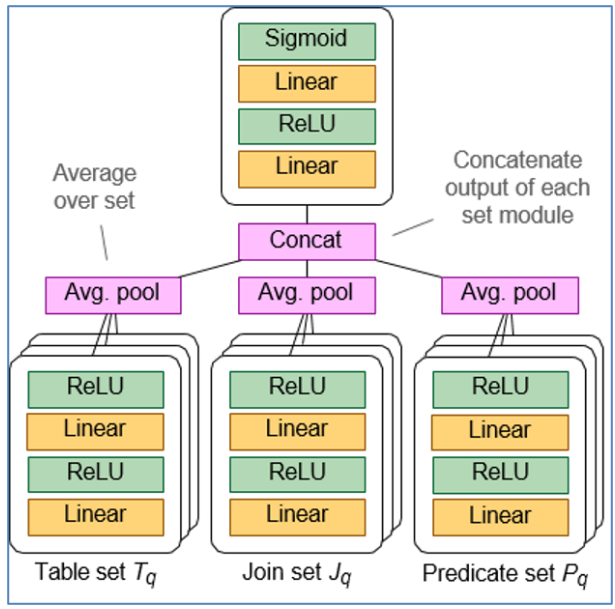


Figure 8.2: `MSCN` Learning Model (Kipf *et al.*, 2019b).

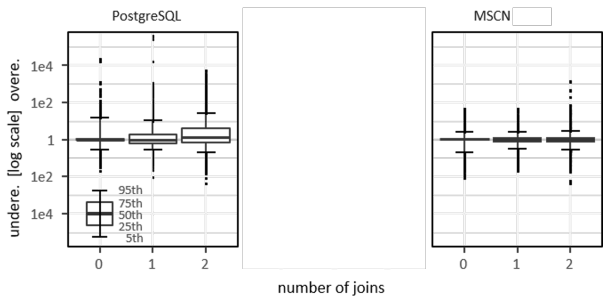


Figure 8.3: `MSCN` Performance (0 – 2 Joins) (Kipf *et al.*, 2019b).

estimation quality becomes noticeably worse, with large Q-error for a significant number of queries. On the other hand, with `MSCN`, the Q-error is largely concentrated around 1, and even among the few outliers, their errors are much lower than `PostgreSQL`.

When we investigate the generalization of this few-join training to queries with more joins, the profile obtained is shown in Figure 8.4. The marked deterioration of `PostgreSQL` is only to be expected given

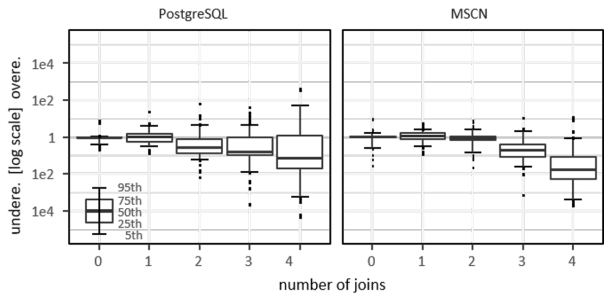


Figure 8.4: `MSCN` Performance (0 – 4 Joins) (Kipf *et al.*, 2019b).

its assumptions of uniformity and independence. However, even `MSCN`, although certainly much better than `PostgreSQL`, shows visible degradation – at 3 joins the 95th percentile Q-error jumps to around 40 and with 4 joins to around 2400!

Further, it is pertinent to note that when Q-error exceeds a factor of 10, any estimator becomes practically useless from an absolute perspective. Therefore, the ability to successfully generalize to unknown queries, especially those encountered in industrial-strength settings, is still an open research question for the learned approaches. And this drawback continues to be the case even with the considerable follow-up literature – while they all improve the average-case performance, the worst-case guarantees still prove to be elusive. In this context, it is appropriate to recall the pathological example in Section 2, which had already hinted at the difficulty faced by summary models in capturing the hyper-sensitive characteristics of database processing.

Turning our attention to training efficiency, the quality versus the number of training epochs is captured in Figure 8.5. We observe that there is an exponential improvement with increasing epochs, and the mean Q-error reaches its steady-state value within 50 epochs.

In summary, while deep learning can certainly help capture complex correlations and address the low-frequency limitations of sampling, these benefits accrue only when there is a good match between training and testing environments. Further, the benefits are in expectation over the training workload, and not on individual queries.

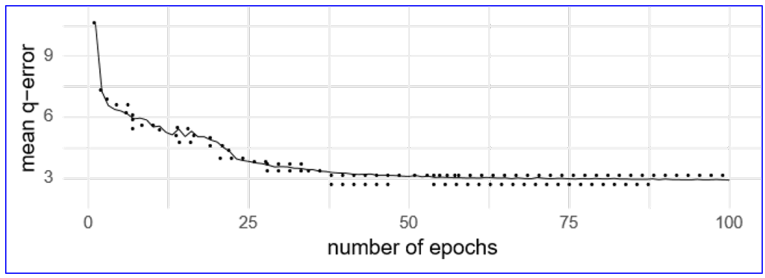


Figure 8.5: Training Convergence Time in `MSCN` (Kipf *et al.*, 2019b).

8.2 Data-based Models

We now turn our attention to the data-based learning models, specifically `NARU` (Neural Relational Understanding). The approach here is to train for learning the joint data distribution of the underlying database using a deep auto-regressive (DAR) model. Specifically, as shown in Figure 8.6, given a source of training tuples, the attributes of each tuple are individually encoded – in the figure, the attributes are x_1, x_2, x_3. And then a likelihood model is created that provides probability distributions at its output. The distributions are such that the first distribution is for the x_1 attribute, whereas the next one is the conditional probability distribution of x_2 given the associated value of x_1, i.e. $P(x_2|x_1)$. And then there is the conditional probability distribution of x_3 given the values of x_1 and x_2, i.e $P(x_3|x_1, x_2)$, and so on. So, essentially, we have a vector of conditional probability distributions, each of which looks at the prior history in the vector sequence to arrive at its own distribution.

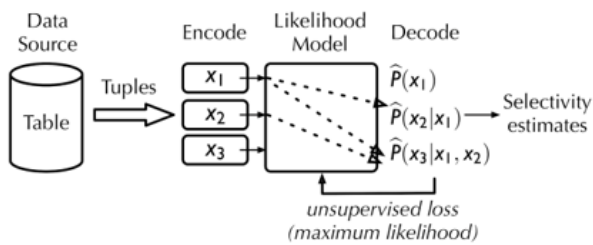


Figure 8.6: Training Phase of `NARU` (Yang *et al.*, 2019).

Based on the above, our goal formally is to learn

$$\mathbf{P}(\mathbf{x}) = \mathbf{\Pi_{i=1}^{n}} \mathbf{P}(\mathbf{x\_i}|\mathbf{x_{<i}}) \tag{8.1}$$

where $\mathbf{x}$ is an n-dimensional tuple. This is achieved through a DAR model, with the convergence for the training leveraging a Maximum Likelihood Estimator. The data tuples are streamed into the DAR model, and the outputs are the conditional probability distributions over columns. The specific choice of DAR is not critical, any of the well-known models such as MADE (Germain *et al.*, 2015), ResMADE (Durkan and Nash, 2019), Transformer (Vaswani *et al.*, 2017), WaveNet (Oord *et al.*, 2016) etc. can be utilized.

Post the training phase, inferencing is carried out as shown in Figure 8.7 – for point queries, the output of the model is directly used, whereas for range queries, a Monte Carlo integration procedure is utilized to estimate the cardinalities.

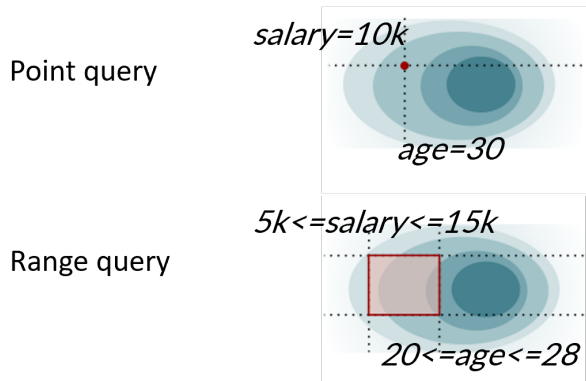


Figure 8.7: Inference Phase of `NARU` (Yang *et al.*, 2019).

Example

To make the above concrete, consider the simple example shown in Figure 8.8 with four rows of *age* and *salary* information. The joint probability of age and salary is given by the values in the `P(A,S)` derived column.

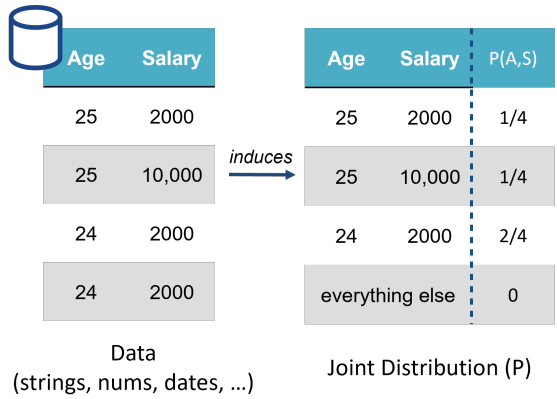


Figure 8.8: Modeling Joint Distribution (Yang *et al.*, 2019).

Now consider the following query:

```
Select * From T Where Age <= 25 and Salary <= 2000
```

The selectivity of this query is mathematically equivalent to its probability density, which is obtained by integrating the joint probability distribution over the predicate validity ranges. Specifically, the selectivity is:

$$Sel(Q) = P(25, 2000) + P(24, 2000) = 1/4 + 2/4 = 0.75$$

These probabilities are computed from the conditional distributions output in Figure 8.6. Essentially, the computation is

$$Sel(Q) = Model(25, 2000) + Model(24, 2000)$$

The important point to note here is that these probability distributions are not materialized because they would be extremely complicated to represent – instead, they are emitted *on demand* by the model.

Further, unlike the traditional histogram model, where the probability of each predicate is considered independent of the others, here the chain-rule factorization ensures there is no information loss. Essentially, we compute $P(A, B, C) = P(B)P(C|B)P(A|C)$ instead of the error-prone $P(A, B, C) = P(A)P(B)P(C)$ formulation.

8.2.1 Range Estimates

The DAR output provides point densities. Therefore, when range predicates are encountered, one obvious way to identify the total range density is to enumerate the valid set of values in each predicate, then take the combination of all such values across the predicates, and finally aggregate the associated point densities. However, this would result in an exponential number of point densities, proportional to the product of the valid domains of the predicates. So, range density computation using this approach is impractical for real-world scenarios. The solution is to instead compute an approximate numerical integration via a Monte Carlo approach.

The standard Monte Carlo approach picks random samples uniformly over the parameter space. However, this is not productive in real-world situations where there is often uneven density and most of the samples do not produce results. The workaround is to selectively give primacy to high point densities and less visibility to low densities. This is achieved via a *progressive sampling* technique, as shown in Figure 8.9. Here, a sample is initially drawn on the valid range of the first dimension, age, of the parameter space. This is followed by conditioning on this sample, and then drawing on the valid range of the second dimension, salary, and so on until the terminal attribute is reached. In the end, appropriate weights are used to return a global density. Essentially, each dimension's sample is used to progressively zoom into the high-density region, and thereby obtain a more accurate estimate.

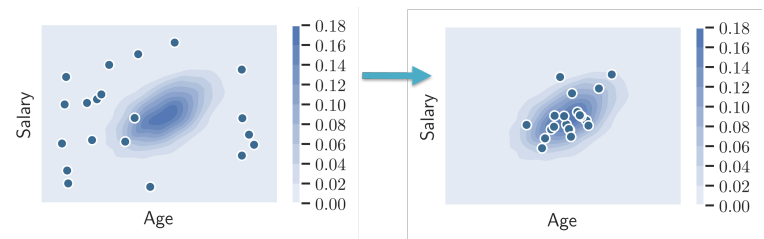


Figure 8.9: Random Sampling vs Progressive Sampling (Yang *et al.*, 2019).

8.2.2 Wildcard Skipping

A practical problem that arises with joint probability distributions is how to handle wild-card (i.e. "don't-care") attributes in a query. An obvious option is to sample over the entire domain of the column, but this would be highly expensive. Instead, Naru takes the approach of introducing a special token during training, called `MASK` (similar to the `ALL` value introduced in the Data Cube framework, Gray *et al.*, 1996), which signifies the *absence* of a column and effectively marginalizes the variable. Specifically, during training, each tuple has its columns randomly perturbed so that the input data also contains the special `MASK` tokens. The query now has its wildcard columns X_i replaced with $X_i = MASK_i$, casting the query into the fully-predicated framework that is amenable to the above-mentioned progressive sampling technique.

8.2.3 Performance

We now move on to profiling Naru's performance. The evaluation was carried out on DMV (State of New York, 2019), a real-world dataset consisting of vehicle registration information in New York. The query workload comprised about 2000 queries, each having between 5 to 11 range and equality predicates. The performance metric is the mean Q-error. A variety of techniques ranging from traditional `PostgreSQL` and a commercial DBMS to sampling-based techniques and supervised learning techniques were compared. The results are presented in Figure 8.10 with the Q-error shown on a log-scale.

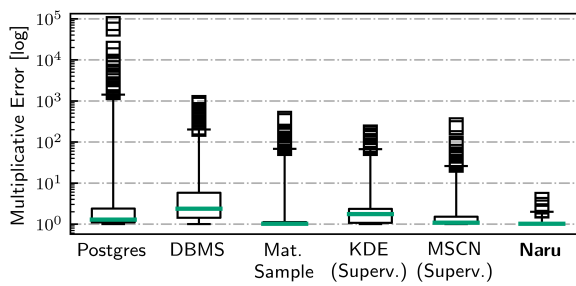


Figure 8.10: Estimation Accuracy of `NARU` (Yang *et al.*, 2019).

We observe a clear progression in the improvement offered by these techniques, with NARU providing the best accuracy. In fact, it improves on the supervised techniques by almost an *order-of-magnitude*. However, even here, the maximum error is close to ten, and therefore the worst-case scenarios continue to be significantly degraded.

Finally, the training time overheads for unsupervised techniques such as NARU are significantly lower compared to their supervised counterparts.

8.3 Unified Models

Encouraged by the successes, especially wrt average-case performance, of query-based and data-based techniques, there have been recent proposals of "unified models" that leverage learning from *both* data and queries. For instance, in Wu and Cong (2021), the gap between data-driven and query-driven methods is closed through a new unified DAR model, called UAE (Unified Autoregressive Estimator), which learns the joint data distribution from both the data and query workload. To enable incorporating the query workload as supervised information in the DAR model, the Gumbel-Softmax technique is leveraged to differentiate the categorically sampled variables. With this enhancement, the DAR can learn joint data distributions directly from queries. Thanks to this hybrid approach, single-digit Q-error values are obtained even at the tail of the error distribution.

8.4 Limitations

Learning certainly does seem a promising and potent approach, especially for modeling correlated joins and complex filter predicates. Notwithstanding, we caution that a variety of concerns still need to be satisfactorily addressed, including:

Universality: The ability to handle unseen queries has not been clearly established.

Explainability: An intuitive explainability of the computed estimates is not provided.

Guarantees: While the average case may be excellent, the worst-case behavior could still be arbitrarily poor.

Heavy-weight: The training phase itself may require very significant computation and collection of data, especially in the supervised techniques.

Uncertainty Estimation: It is hard to quantify the risk involved in trusting the model.

As further quantitative evidence of the above limitations, a recently published paper by Lehmann *et al.* (2023) titled "Is Your Learned Query Optimizer Behaving As You Expect: A Machine Learning Perspective", concludes after a comprehensive empirical evaluation that even the best ML approaches available today do not systematically outperform the classical `PostgreSQL` optimizer!

9

Holistic Robustness

Over the past sections, we have covered a variety of techniques for infusing robustness into the query optimization and processing framework. However, they have been discussed in isolation, and a natural question that one could ask is whether the various techniques could be holistically combined in a complementary manner.

The good news is that we can bring them together, as shown in the architectural diagram of Figure 9.1, which is at the level of complete plans. Specifically, at the base, we construct a calibrated cost model, which is used to identify the isocost contours of the robust plan algorithms. But in this cost model, the calibration is done on Q-error instead of the relative error used in the original formulation. Second, we use the progressive sampling technique of Naru to derive higher-quality estimates. Then, we employ the `CostGreedy` reduction algorithm to ensure that the number of plans in each of the contours is limited to a small number. And finally, we use these contours as inputs to the `SpillBound` approach to ensure worst-case performance guarantees on individual queries.

Within each plan, we use the robust operator techniques that were discussed in Section 3, as shown in Figure 9.2. Specifically, `SmoothScan`

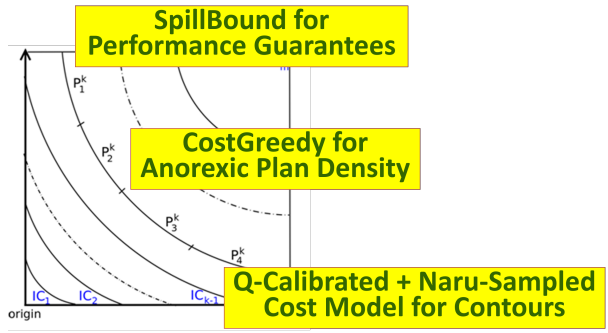


Figure 9.1: RQP Architecture Plan-level.

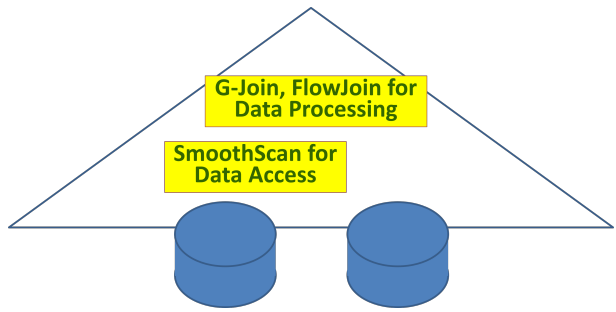


Figure 9.2: RQP Architecture Intra-plan.

replaces the sequential scan and index scan for data access, and `G-Join` provides a unified join algorithm in centralized databases. Further, although not discussed in this monograph, the `FlowJoin` algorithm (Rödiger *et al.*, 2016) can be used to handle distributed contexts.

Overall, while absolute robustness may always remain a dream, it certainly appears feasible to achieve a practical level with the tools and techniques that are currently under our command.

10

Future Research Directions

From the presentation thus far, it should be evident that significant progress has been achieved over the past decade on providing robustness, with regard to both processing of workloads and individual queries. Notwithstanding, there still remain a variety of challenging research problems whose solutions could materially improve the robustness profiles.

Join-Graph-Sensitive Robustness Thus far, we have essentially considered SQL queries as monoliths and not really given heed to their internal structures. However, in practice, most database queries have join-graphs with special topologies such as *chain*, *star*, *cycle*, etc. And leveraging this structure, which is known at compile-time, could potentially provide improved robustness guarantees.

As a case in point, it has been shown in Kumar (2018) that the MSO guarantees of `SpillBound` can be improved from the quadratic $D^2 + 3D$ for general join-graphs to a linear $8D - 6$ for chain queries! The MSO gap between star queries and chain queries is quantitatively captured in Figure 10.1 as a function of the number of edges in the query join-graph.

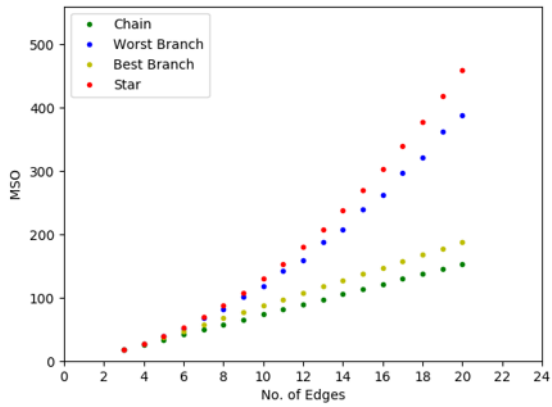


Figure 10.1: MSO Guarantees based on Query Graph (Kumar, 2018).

It would be interesting to extend the above study and develop MSO guarantees for other common join-graph topologies.

Graceful Performance Degradation A major problem faced in real deployments are "performance cliffs", where the performance suddenly degrades precipitously although there has only been a minor change in the operational environment. This is particularly true with regard to hardware resources, such as memory. So, an important future challenge is to design algorithms that *provably* degrade gracefully with regard to all their performance-related parameters.

In fact, even with `PlanBouquet`-style query processing strategies, there are performance jumps every time a contour is crossed. For instance, in the 1D case with a cost-doubling regime in place, the sub-optimality falls from 4 to 3 at each contour crossing, and then works its way back to 4 until just before the next crossing. However, a perhaps more appropriate view is that these jumps are "walls", and not cliffs, since the jump magnitude is both bounded and limited. Nevertheless, it would be an intellectually appealing exercise to investigate whether, in principle, a smooth version of `PlanBouquet` could be designed.

Refined Cost Model Calibration The cost calibration model discussed in Section 7 took the `PostgreSQL` basic 5-parameter model as a given

for the entire suite of operators. And the same calibrated values for these parameters were used across all operators. However, by incorporating operator-specific features and operator-specific calibration of the associated coefficients, we could potentially reap enhanced accuracy for cost modeling. That is, to move from a global cost model to locally tuned cost models, while retaining the benefits of the calibration methodology.

Robustness Benchmarks A pre-requisite for confirming the robustness offered by new approaches are principled benchmarks that exercise and push the system to its limits. This is critical since standard benchmarks, such as TPC-DS, measure performance, not robustness. Some recent efforts in this direction include **OptMark** (Li *et al.*, 2016), **JOB** (Leis *et al.*, 2018) and **OTT** (Wu *et al.*, 2016). However, there remain several aspects of robustness that are yet to be covered. For instance, we would ideally wish to have non-pathological realistic benchmarks that highlight issues such as performance cliffs.

Machine Learning Techniques for Component Selection We advocate the database engine to host a *multiplicity* of alternative components for a given task, with the intention of separately but cooperatively catering to the various query processing environments. For instance, if the cardinality estimates are expected to be reasonably accurate, then the native query optimizer is appropriate to choose the execution plan. On the other hand, if the estimates are expected to be brittle, then robust techniques can be invoked instead. An obvious question that arises with such an architecture is determining the specific environment that is currently operational, and hence the associated component choice. Machine learning techniques could be used to make this selection, similar to the exercise recently carried out in Hüske (2016) in the context of analytical data flows.

Dimensionality Reduction of the Selectivity Space An important design question in the RQP framework is to identify the query predicates that constitute the error-prone selectivity space. A simple conservative option would be to consider all predicates (filter, join, projection) to

be potentially uncertain, but this would needlessly increase the MSO guarantees due to the quadratic dependency on D, the dimensionality of the space. Therefore, a fruitful exercise would be to use a combination of domain knowledge, query logs, cost behavior and machine learning techniques to restrict the dimensions to include only those expected to suffer error-prone estimations. A preliminary study in this direction is discussed in Purandare *et al.* (2018).

Cost Models for Upcoming Architectures In the discussion thus far, we had assumed a traditional on-premise computational environment, usually referred to as a shared-nothing architecture, accompanied by single-threaded execution. However, in recent times, disaggregated resource architectures have been gaining popularity, especially for Cloud-resident data warehouses. While the cardinality model, which is logical, is largely unaffected, there may be greater variance in cost modeling quality due to the attendant complexity of distributed control. Moreover, users may leverage the flexibility offered by the Cloud to dynamically assemble query execution platforms, potentially requiring the operator cost model to be constructed on-the-fly.

A similar challenge could arise in the context of massively parallel execution engines as well. While the optimizer's plan choices are likely to change – for instance, bushy operator trees are now more likely to find favour – again it is the cost model that is likely to be primarily affected by the platform. Therefore, developing accurate and fast methodologies for cost modeling of these contemporary architectures would be a highly relevant research study.

10.1 Closing Note

Robustness is a highly relevant but poorly understood aspect of declarative query processing in relational database systems. In this monograph, we have attempted to shed light on various mechanisms through which robustness can be incorporated in these systems. It is our hope that the combination of theoretical and empirical innovations outlined here will encourage the research community to revisit this classical field with renewed vigor, eventually leading to "bullet-proof" database engines.

11

Additional Reading

The references include publications, beyond those directly referenced in the monograph, related to robust query processing.

Acknowledgements

I am deeply grateful to my home institution, the Indian Institute of Science, for providing a supportive research environment. My profound appreciation goes out to the several graduate students who have worked with me over the years on this monograph's topic, especially my doctoral scholars, Anshuman Dutt and Srinivas Karthik, and master's advisees, Harish Doraiswamy and Pooja Darera. Many thanks to fellow database researchers – Renata Borovica-Gajic, Goetz Graefe, Andreas Kipf, Thomas Neumann, Wolf Roediger, Wentao Wu, Zongheng Yang – who generously provided the presentation material related to their work described here. Their support greatly facilitated my RQP tutorials at ICDE 2019 and VLDB 2020, which provided the core material for this monograph. My deep gratitude also to the reviewers of the original manuscript for their careful reading and perceptive comments, which resulted in a visibly improved end product. Finally, I am much obliged to Surajit Chaudhuri for persistently motivating me over the past year to embark on and complete this ambitious writing exercise.

References

Aboulnaga, A. and S. Chaudhuri. (1999). "Self-tuning histograms: building histograms without looking at data". In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data. SIGMOD '99*. Philadelphia, Pennsylvania, USA: Association for Computing Machinery. 181–192. DOI: 10.1145/304182.304198.

Akdere, M., U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik. (2012). "Learning-based Query Performance Modeling and Prediction". In: *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*. Ed. by A. Kementsietsidis and M. A. V. Salles. IEEE Computer Society. 390–401. DOI: 10.1109/ICDE.2012.64.

Avnur, R. and J. M. Hellerstein. (2000). "Eddies: Continuously Adaptive Query Processing". In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. SIGMOD '00*. Dallas, Texas, USA: Association for Computing Machinery. 261–272. DOI: 10.1145/342009.335420.

Babcock, B. and S. Chaudhuri. (2005). "Towards a Robust Query Optimizer: A Principled and Practical Approach". In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. SIGMOD '05*. Baltimore, Maryland: Association for Computing Machinery. 119–130. DOI: 10.1145/1066157.1066172.

Babu, S., P. Bizarro, and D. DeWitt. (2005). "Proactive Re-Optimization". In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. SIGMOD '05*. Baltimore, Maryland: Association for Computing Machinery. 107–118. DOI: 10.1145/1066157.1066171.

Bizarro, P., N. Bruno, and D. J. DeWitt. (2009). "Progressive Parametric Query Optimization". *IEEE Transactions on Knowledge and Data Engineering*. 21(04): 582–594. DOI: 10.1109/TKDE.2008.160.

Böhm, A., M. Christakis, E. Lo, and M. Rigger. (2021). "Ensuring the Reliability and Robustness of Database Management Systems (Dagstuhl Seminar 21442)". *Dagstuhl Reports*. 11(10): 20–35. DOI: 10.4230/DAGREP.11.10.20.

Borovica-Gajic, R., G. Graefe, and A. W. Lee. (2017). "Robust Performance in Database Query Processing (Dagstuhl Seminar 17222)". *Dagstuhl Reports*. 7(5): 169–180. DOI: 10.4230/DAGREP.7.5.169.

Borovica-Gajic, R., S. Idreos, A. Ailamaki, M. Zukowski, and C. Fraser. (2015). "Smooth Scan: Statistics-oblivious access paths". In: *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*. Ed. by J. Gehrke, W. Lehner, K. Shim, S. K. Cha, and G. M. Lohman. IEEE Computer Society. 315–326. DOI: 10.1109/ICDE.2015.7113294.

Borovica-Gajic, R., S. Idreos, A. Ailamaki, M. Zukowski, and C. Fraser. (2018). "Smooth Scan: robust access path selection without cardinality estimation". *VLDB J.* 27(4): 521–545. DOI: 10.1007/S00778-018-0507-8.

Chaudhuri, S. (1998). "An Overview of Query Optimization in Relational Systems". In: *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*. Ed. by A. O. Mendelzon and J. Paredaens. ACM Press. 34–43. DOI: 10.1145/275487.275492.

Chaudhuri, S. (2009). "Query optimizers: time to rethink the contract?" In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*. Ed. by U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul. ACM. 961–968. DOI: 10.1145/1559845.1559955.

Chaudhuri, S., H. Lee, and V. R. Narasayya. (2010). "Variance Aware Optimization of Parameterized Queries". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. SIGMOD '10*. Indianapolis, Indiana, USA: Association for Computing Machinery. 531–542. DOI: 10.1145/1807167.1807226.

Chaudhuri, S., V. Narasayya, and R. Ramamurthy. (2008). "A Pay-as-You-Go Framework for Query Execution Feedback". *Proc. VLDB Endow.* 1(1): 1141–1152. DOI: 10.14778/1453856.1453977.

Chrobak, M., C. Kenyon, J. Noga, and N. E. Young. (2008). "Incremental Medians via Online Bidding". *Algorithmica.* 50(4): 455–478. DOI: 10.1007/S00453-007-9005-X.

Chu, F., J. Halpern, and J. Gehrke. (2002). "Least Expected Cost Query Optimization: What Can We Expect?" In: *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. PODS '02*. New York, NY, USA: Association for Computing Machinery. 293–302. DOI: 10.1145/543613.543651.

Chu, F. C., J. Y. Halpern, and P. Seshadri. (1999). "Least Expected Cost Query Optimization: An Exercise in Utility". In: *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*. Ed. by V. Vianu and C. H. Papadimitriou. ACM Press. 138–147. DOI: 10.1145/303976.303990.

Cole, R. L. and G. Graefe. (1994). "Optimization of Dynamic Query Evaluation Plans". In: *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data. SIGMOD '94*. Minneapolis, Minnesota, USA: Association for Computing Machinery. 150–160. DOI: 10.1145/191839.191872.

Deshpande, A., Z. Ives, and V. Raman. (2007). "Adaptive Query Processing". *Found. Trends databases.* 1(Jan.): 1–140. DOI: 10.1561/1900000001.

Doraiswamy, H., P. N. Darera, and J. R. Haritsa. (2007). "On the Production of Anorexic Plan Diagrams". In: *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*. Ed. by C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C. Kanne, W. Klas, and E. J. Neuhold. ACM. 1081–1092. URL: www.vldb.org/conf/2007/papers/research/p1081-d.pdf.

Doraiswamy, H., P. N. Darera, and J. R. Haritsa. (2008). "Identifying robust plans through plan diagram reduction". *Proc. VLDB Endow.* 1(1): 1124–1140. DOI: 10.14778/1453856.1453976.

Durkan, C. and C. Nash. (2019). "Autoregressive Energy Machines". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. *Proceedings of Machine Learning Research*. PMLR. 1735–1744. URL: proceedings.mlr.press/v97/durkan19a.html.

Dutt, A. and J. R. Haritsa. (2016). "Plan Bouquets: A Fragrant Approach to Robust Query Processing". *ACM Trans. Database Syst.* 41(2): 11:1–11:37. DOI: 10.1145/2901738.

Dutt, A. and J. R. Haritsa. (2018). "Query Processing without Estimation". *Tech. rep.* No. TR-2014-01. DSL/SERC, Indian Institute of Science. URL: dsl.cds.iisc.ac.in/publications/report/TR/TR-2014-01.pdf.

Dutt, A., V. R. Narasayya, and S. Chaudhuri. (2017). "Leveraging Recosting for Online Optimization of Parameterized Queries with Guarantees". In: *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. Ed. by S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciu. ACM. 1539–1554. DOI: 10.1145/3035918.3064040.

Dutt, A., C. Wang, V. R. Narasayya, and S. Chaudhuri. (2020). "Efficiently Approximating Selectivity Functions using Low Overhead Regression Models". *Proc. VLDB Endow.* 13(11): 2215–2228. URL: www.vldb.org/pvldb/vol13/p2215-dutt.pdf.

Dutt, A., C. Wang, A. Nazi, S. Kandula, V. R. Narasayya, and S. Chaudhuri. (2019). "Selectivity Estimation for Range Predicates using Lightweight Models". *Proc. VLDB Endow.* 12(9): 1044–1057. DOI: 10.14778/3329772.3329780.

Ganapathi, A., H. A. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. A. Patterson. (2009). "Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning". In: *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China.* Ed. by Y. E. Ioannidis, D. L. Lee, and R. T. Ng. IEEE Computer Society. 592–603. DOI: 10.1109/ICDE.2009.130.

Garey, M. R. and D. S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences).* First Edition. W. H. Freeman. URL: www.amazon.com/Computers-Intractability-NP-Completeness-Mathematical-Sciences/dp/0716710455.

Germain, M., K. Gregor, I. Murray, and H. Larochelle. (2015). "MADE: Masked Autoencoder for Distribution Estimation". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015.* Ed. by F. R. Bach and D. M. Blei. Vol. 37. *JMLR Workshop and Conference Proceedings.* JMLR.org. 881–889. URL: proceedings.mlr.press/v37/germain15.html.

Graefe, G. (2012). "New algorithms for join and grouping operations". *Comput. Sci. Res. Dev.* 27(1): 3–27. DOI: 10.1007/S00450-011-0186-9.

Graefe, G., W. Guy, H. A. Kuno, and G. N. Paulley. (2012). "Robust Query Processing (Dagstuhl Seminar 12321)". *Dagstuhl Reports.* 2(8): 1–15. DOI: 10.4230/DAGREP.2.8.1.

Graefe, G., A. C. König, H. A. Kuno, V. Markl, and K. Sattler. (2010). "Robust Query Processing (Dagstuhl Seminar 10381)". Dagstuhl Seminar Proceedings 10381. URL: drops.dagstuhl.de/portals/10381/.

Gray, J., A. Bosworth, A. Layman, and H. Pirahesh. (1996). "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total". In: *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana, USA*. Ed. by S. Y. W. Su. IEEE Computer Society. 152–159. DOI: 10.1109/ICDE.1996.492099.

Haas, P. J., J. F. Naughton, S. Seshadri, and A. N. Swami. (1996). "Selectivity and Cost Estimation for Joins Based on Random Sampling". *J. Comput. Syst. Sci.* 52(3): 550–569. DOI: 10.1006/JCSS.1996.0041.

Han, Y., Z. Wu, P. Wu, R. Zhu, J. Yang, L. W. Tan, K. Zeng, G. Cong, Y. Qin, A. Pfadler, Z. Qian, J. Zhou, J. Li, and B. Cui. (2021). "Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation". *Proc. VLDB Endow.* 15(4): 752–765. DOI: 10.14778/3503585.3503586.

Harmouch, H. and F. Naumann. (2017). "Cardinality Estimation: An Experimental Survey". *Proc. VLDB Endow.* 11(4): 499–512. DOI: 10.1145/3186728.3164145.

Hasan, S., S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das. (2020). "Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. Ed. by D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo. ACM. 1035–1050. DOI: 10.1145/3318464.3389741.

Havenstein, D., P. Lysakovski, N. May, G. Moerkotte, and G. Steidl. (2020). "Fast Entropy Maximization for Selectivity Estimation of Conjunctive Predicates on CPUs and GPUs". In: *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*. Ed. by A. Bonifati, Y. Zhou, M. A. V. Salles, A. Böhm, D. Olteanu, G. H. L. Fletcher, A. Khan, and B. Yang. OpenProceedings.org. 546–554. DOI: 10.5441/002/EDBT.2020.65.

Hayek, R. and O. Shmueli. (2020a). "Improved Cardinality Estimation by Learning Queries Containment Rates". In: *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*. Ed. by A. Bonifati, Y. Zhou, M. A. V. Salles, A. Böhm, D. Olteanu, G. H. L. Fletcher, A. Khan, and B. Yang. OpenProceedings.org. 157–168. DOI: 10.5441/002/EDBT.2020.15.

Hayek, R. and O. Shmueli. (2020b). "NN-based Transformation of Any SQL Cardinality Estimator for Handling DISTINCT, AND, OR and NOT". *CoRR*. abs/2004.07009. arXiv: 2004.07009. URL: arxiv.org/abs/2004.07009.

Heimel, M., M. Kiefer, and V. Markl. (2015). "Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. Ed. by T. K. Sellis, S. B. Davidson, and Z. G. Ives. ACM. 1477–1492. DOI: 10.1145/2723372.2749438.

Hilprecht, B., A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. (2020). "DeepDB: Learn from Data, not from Queries!" *Proc. VLDB Endow.* 13(7): 992–1005. DOI: 10.14778/3384345.3384349.

Hulgeri, A. and S. Sudarshan. (2002). "Parametric Query Optimization for Linear and Piecewise Linear Cost Functions". In: *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002*. Morgan Kaufmann. 167–178. DOI: 10.1016/B978-155860869-6/50023-8.

Hüske, F. (2016). "Specification and optimization of analytical data flows". *PhD thesis*. Technical University of Berlin, Germany. URL: nbn-resolving.org/urn:nbn:de:101:1-201804165046.

Ioannidis, Y. E. and S. Christodoulakis. (1991). "On the Propagation of Errors in the Size of Join Results". In: *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data. SIGMOD '91*. Denver, Colorado, USA: Association for Computing Machinery. 268–277. DOI: 10.1145/115790.115835.

Jahangiri, S., M. J. Carey, and J.-C. Freytag. (2022). "Design trade-offs for a robust dynamic hybrid hash join". *Proc. VLDB Endow.* 15(10): 2257–2269. DOI: 10.14778/3547305.3547327.

Kabra, N. and D. J. DeWitt. (1998a). "Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans". In: *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*. Ed. by L. M. Haas and A. Tiwary. ACM Press. 106–117. DOI: 10.1145/276304.276315.

Kabra, N. and D. J. DeWitt. (1998b). "Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans". In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data. SIGMOD '98*. Seattle, Washington, USA: Association for Computing Machinery. 106–117. DOI: 10.1145/276304.276315.

Karthik, S., J. R. Haritsa, S. Kenkre, and V. Pandit. (2016). "Platform-independent robust query processing". In: *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*. IEEE Computer Society. 325–336. DOI: 10.1109/ICDE.2016.7498251.

Karthik, S., J. R. Haritsa, S. Kenkre, V. Pandit, and L. Krishnan. (2019). "Platform-Independent Robust Query Processing". *IEEE Trans. Knowl. Data Eng.* 31(1): 17–31. DOI: 10.1109/TKDE.2017.2664827.

Kiefer, M., M. Heimel, S. Breß, and V. Markl. (2017). "Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models". *Proc. VLDB Endow.* 10(13): 2085–2096. DOI: 10.14778/3151106.3151112.

Kipf, A., M. Freitag, D. Vorona, P. Boncz, T. Neumann, and A. Kemper. (2019a). "Estimating Filtered Group-By Queries is Hard: Deep Learning to the Rescue". In: *Proc of VLDB AIDB Workshop*. URL: drive.google.com/file/d/1Yt2w7_8UKFxsgcnWE8BjnFwhw509lyyY/view.

Kipf, A., T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. (2019b). "Learned Cardinalities: Estimating Correlated Joins with Deep Learning". In: *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. URL: cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf.

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. USA: Addison-Wesley Longman Publishing Co., Inc.

Kreyszig, E., H. Kreyszig, and E. J. Norminton. (2011). *Advanced Engineering Mathematics*. Tenth. Hoboken, NJ: Wiley.

Krishnan, S., Z. Yang, K. Goldberg, J. M. Hellerstein, and I. Stoica. (2018). "Learning to Optimize Join Queries With Deep Reinforcement Learning". *CoRR*. abs/1808.03196. arXiv: 1808.03196. URL: arxiv.org/abs/1808.03196.

Kumar, G. (2018). "Sub-query Based Approach for Robust Query Processing". *ME thesis*. CSA, IISc. URL: dsl.cds.iisc.ac.in/publications/thesis/gourav18.pdf.

Lehmann, C., P. Sulimov, and K. Stockinger. (2023). "Is Your Learned Query Optimizer Behaving As You Expect? A Machine Learning Perspective". DOI: 10.48550/ARXIV.2309.01551. arXiv: 2309.01551.

Leis, V., A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann. (2015). "How Good Are Query Optimizers, Really?" *Proc. VLDB Endow.* 9(3): 204–215. DOI: 10.14778/2850583.2850594.

Leis, V., B. Radke, A. Gubichev, A. Kemper, and T. Neumann. (2017). "Cardinality Estimation Done Right: Index-Based Join Sampling". In: *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org. URL: cidrdb.org/cidr2017/papers/p9-leis-cidr17.pdf.

Leis, V., B. Radke, A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann. (2018). "Query optimization through the looking glass, and what we found running the Join Order Benchmark". *VLDB J.* 27(5): 643–668. DOI: 10.1007/S00778-017-0480-7.

Li, G. (2010). "On the Design and Evaluation of a New Order-Based Join Algorithm". *MS thesis*. URL: asterix.ics.uci.edu/thesis/Guangqiang_Li_MS_thesis_2010.pdf.

Li, K. and G. Li. (2018). "Approximate Query Processing: What is New and Where to Go? - A Survey on Approximate Query Processing". *Data Sci. Eng.* 3(4): 379–397. DOI: 10.1007/S41019-018-0074-4.

Li, Z., O. Papaemmanouil, and M. Cherniack. (2016). "OptMark: A Toolkit for Benchmarking Query Optimizers". In: *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*. Ed. by S. Mukhopadhyay, C. Zhai, E. Bertino, F. Crestani, J. Mostafa, J. Tang, L. Si, X. Zhou, Y. Chang, Y. Li, and P. Sondhi. ACM. 2155–2160. DOI: 10.1145/2983323.2983658.

Liu, H., M. Xu, Z. Yu, V. Corvinelli, and C. Zuzarte. (2015). "Cardinality estimation using neural networks". In: *Proceedings of 25th Annual International Conference on Computer Science and Software Engineering, CASCON 2015, Markham, Ontario, Canada, 2-4 November, 2015*. Ed. by J. Gould, M. Litoiu, and H. Lutfiyya. IBM / ACM. 53–59. URL: dl.acm.org/citation.cfm?id=2886453.

Lohman, G. (2014). "Is Query Optimization a Solved Problem?" URL: wp.sigmod.org/?p=1075.

Lu, Y., S. Kandula, A. C. König, and S. Chaudhuri. (2021). "Pre-training Summarization Models of Structured Datasets for Cardinality Estimation". *Proc. VLDB Endow.* 15(3): 414–426. DOI: 10.14778/3494124.3494127.

Ma, L., B. Ding, S. Das, and A. Swaminathan. (2020). "Active Learning for ML Enhanced Database Systems". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. Ed. by D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo. ACM. 175–191. DOI: 10.1145/3318464.3389768.

Malik, T., R. C. Burns, and N. V. Chawla. (2007). "A Black-Box Approach to Query Cardinality Estimation". In: *Third Biennial Conference on Innovative Data Systems Research, CIDR 2007, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*. www.cidrdb.org. 56–67. URL: cidrdb.org/cidr2007/papers/cidr07p06.pdf.

Marcus, R., P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul. (2019). "Neo: A Learned Query Optimizer". *Proc. VLDB Endow.* 12(11): 1705–1718. DOI: 10.14778/3342263.3342644.

Marcus, R. and O. Papaemmanouil. (2019). "Towards a Hands-Free Query Optimizer through Deep Learning". In: *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. URL: cidrdb.org/cidr2019/papers/p96-marcus-cidr19.pdf.

Markl, V., V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic. (2004). "Robust Query Processing through Progressive Optimization". In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. SIGMOD '04*. Paris, France: Association for Computing Machinery. 659–670. DOI: 10.1145/1007568.1007642.

Matias, Y., J. S. Vitter, and M. Wang. (1998). "Wavelet-Based Histograms for Selectivity Estimation". In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data. SIGMOD '98*. Seattle, Washington, USA: Association for Computing Machinery. 448–459. DOI: 10.1145/276304.276344.

Mattig, M., T. Fober, C. Beilschmidt, and B. Seeger. (2018). "Kernel-Based Cardinality Estimation on Metric Data". In: *Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*. Ed. by M. H. Böhlen, R. Pichler, N. May, E. Rahm, S. Wu, and K. Hose. OpenProceedings.org. 349–360. DOI: 10.5441/002/EDBT.2018.31.

Moerkotte, G., T. Neumann, and G. Steidl. (2009). "Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors". *Proc. VLDB Endow.* 2(1): 982–993. DOI: 10.14778/1687627.1687738.

Müller, M., G. Moerkotte, and O. Kolb. (2018). "Improved Selectivity Estimation by Combining Knowledge from Sampling and Synopses". *Proc. VLDB Endow.* 11(9): 1016–1028. URL: www.vldb.org/pvldb/vol11/p1016-muller.pdf.

Muralikrishna, M. and D. J. DeWitt. (1988). "Equi-Depth Multidimensional Histograms". In: *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data. SIGMOD '88*. Chicago, Illinois, USA: Association for Computing Machinery. 28–36. DOI: 10.1145/50202.50205.

Negi, P., R. Marcus, H. Mao, N. Tatbul, T. Kraska, and M. Alizadeh. (2020). "Cost-Guided Cardinality Estimation: Focus Where it Matters". In: *36th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE. 154–157. DOI: 10.1109/ICDEW49219.2020.00034.

Neumann, T. and C. A. Galindo-Legaria. (2013). "Taking the Edge off Cardinality Estimation Errors using Incremental Execution". In: *Datenbanksysteme für Business, Technologie und Web (BTW), 15. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 11.-15.3.2013 in Magdeburg, Germany. Proceedings*. Ed. by V. Markl, G. Saake, K. Sattler, G. Hackenbroich, B. Mitschang, T. Härder, and V. Köppen. Vol. P-214. *LNI*. GI. 73–92. URL: https://dl.gi.de/handle/20.500.12116/17356.

Neumann, T. and B. Radke. (2018). "Adaptive Optimization of Very Large Join Queries". In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. Ed. by G. Das, C. M. Jermaine, and P. A. Bernstein. ACM. 677–692. DOI: 10.1145/3183713.3183733.

Ngo, H. Q., C. Ré, and A. Rudra. (2014). "Skew strikes back: new developments in the theory of join algorithms". *SIGMOD Rec.* 42(4): 5–16. DOI: 10.1145/2590989.2590991.

Ngo, H. Q. (2018). "Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems". In: *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. PODS '18*. Houston, TX, USA: Association for Computing Machinery. 111–124. DOI: 10.1145/3196959.3196990.

Oord, A. van den, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. (2016). "WaveNet: A Generative Model for Raw Audio". In: *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*. ISCA. 125. URL: www.isca-speech.org/archive/SSW%5C_2016/abstracts/ssw9%5C_DS-4%5C_van%5C_den%5C_Oord.html.

Ortiz, J., M. Balazinska, J. Gehrke, and S. S. Keerthi. (2018). "Learning State Representations for Query Optimization with Deep Reinforcement Learning". In: *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning, DEEM@SIGMOD 2018, Houston, TX, USA, June 15, 2018*. Ed. by S. Schelter, S. Seufert, and A. Kumar. ACM. 4:1–4:4. DOI: 10.1145/3209889.3209890.

Ortiz, J., M. Balazinska, J. Gehrke, and S. S. Keerthi. (2019). "An Empirical Analysis of Deep Learning for Cardinality Estimation". *CoRR*. abs/1905.06425. arXiv: 1905.06425.

Ouared, A., A. Chadli, and M. A. Daoud. (2022). "DeepCM: Deep neural networks to improve accuracy prediction of database cost models". *Concurr. Comput. Pract. Exp.* 34(10). DOI: 10.1002/CPE.6724.

Parameswaran, A. G. (2012). "An interview with Surajit Chaudhuri". *XRDS*. 19(1): 38–39. DOI: 10.1145/2331042.2331055.

Park, Y., S. Zhong, and B. Mozafari. (2020). "QuickSel: Quick Selectivity Learning with Mixture Models". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. Ed. by D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo. ACM. 1017–1033. DOI: 10.1145/3318464.3389727.

Polyzotis, N. (2005). "Selectivity-Based Partitioning: A Divide-and-Union Paradigm for Effective Query Optimization". In: *Proceedings of the 14th ACM International Conference on Information and Knowledge Management. CIKM '05*. Bremen, Germany: Association for Computing Machinery. 720–727. DOI: 10.1145/1099554.1099730.

Purandare, S., S. Karthik, and J. R. Haritsa. (2018). "Dimensionality Reduction Techniques for Robust Query Processing". *Tech. rep.* No. TR-2018-02. DSL/CDS, Indian Institute of Science. URL: dsl.cds.iisc.ac.in/publications/report/TR/TR-2018-02.pdf.

Ramachandra, K., K. Park, K. V. Emani, A. Halverson, C. A. Galindo-Legaria, and C. Cunningham. (2017). "Froid: Optimization of Imperative Programs in a Relational Database". *Proc. VLDB Endow.* 11(4): 432–444. DOI: 10.1145/3186728.3164140.

Reddy, N. and J. R. Haritsa. (2005). "Analyzing Plan Diagrams of Database Query Optimizers". In: *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*. Ed. by K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P. Larson, and B. C. Ooi. ACM. 1228–1240. URL: www.vldb.org/archives/website/2005/program/paper/fri/p1228-reddy.pdf.

Rödiger, W., S. Idicula, A. Kemper, and T. Neumann. (2016). "Flow-Join: Adaptive skew handling for distributed joins over high-speed networks". In: *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*. IEEE Computer Society. 1194–1205. DOI: 10.1109/ICDE.2016.7498324.

Selinger, P. G., M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. (1979). "Access Path Selection in a Relational Database Management System". In: *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data. SIGMOD '79*. Boston, Massachusetts: Association for Computing Machinery. 23–34. DOI: 10.1145/582095.582099.

Silberschatz, A., H. F. Korth, and S. Sudarshan. (2020). *Database System Concepts, Seventh Edition*. McGraw-Hill Book Company. URL: https://www.db-book.com/.

State of New York. (2019). *Vehicle, snowmobile, and boat registrations*. URL: catalog.data.gov/dataset/vehicle-snowmobile-and-boat-registrations.

Stillger, M., G. M. Lohman, V. Markl, and M. Kandil. (2001). "LEO - DB2's LEarning Optimizer". In: *Proceedings of the 27th International Conference on Very Large Data Bases. VLDB '01*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 19–28.

Sun, J. and G. Li. (2019). "An End-to-End Learning-based Cost Estimator". *Proc. VLDB Endow.* 13(3): 307–319. DOI: 10.14778/3368289.3368296.

Sun, J., J. Zhang, Z. Sun, G. Li, and N. Tang. (2021). "Learned Cardinality Estimation: A Design Space Exploration and A Comparative Evaluation". *Proc. VLDB Endow.* 15(1): 85–97. DOI: 10.14778/3485450.3485459.

Sun, L., C. Li, T. Ji, and H. Chen. (2023). "MOSE: A Monotonic Selectivity Estimator Using Learned CDF". *IEEE Trans. Knowl. Data Eng.* 35(3): 2823–2836. DOI: 10.1109/TKDE.2021.3112753.

Swami, A. (1989a). "Optimization of large join queries: combining heuristics and combinatorial techniques". In: *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data. SIGMOD '89*. Portland, Oregon, USA: Association for Computing Machinery. 367–376. DOI: 10.1145/67544.66961.

Swami, A. (1989b). "Optimization of large join queries: combining heuristics and combinatorial techniques". *SIGMOD Rec.* 18(2): 367–376. DOI: 10.1145/66926.66961.

Transaction Processing Council. (2024a). *TPC-DS*. URL: www.tpc.org/tpcds.

Transaction Processing Council. (2024b). *TPC-H*. URL: www.tpc.org/tpch.

Tzoumas, K., A. Deshpande, and C. S. Jensen. (2011). "Lightweight Graphical Models for Selectivity Estimation Without Independence Assumptions". *Proc. VLDB Endow.* 4(11): 852–863. URL: www.vldb.org/pvldb/vol4/p852-tzoumas.pdf.

Tzoumas, K., A. Deshpande, and C. S. Jensen. (2013). "Efficiently adapting graphical models for selectivity estimation". *VLDB J.* 22(1): 3–27. DOI: 10.1007/S00778-012-0293-7.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett. 5998–6008. URL: proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Venkatesh, S. K., J. R. Haritsa, S. Kenkre, and V. Pandit. (2018). "A Concave Path to Low-overhead Robust Query Processing". *Proc. VLDB Endow.* 11(13): 2183–2195. DOI: 10.14778/3275366.3275368.

Wang, X., C. Qu, W. Wu, J. Wang, and Q. Zhou. (2021a). "Are We Ready For Learned Cardinality Estimation?" *Proc. VLDB Endow.* 14(9): 1640–1654. DOI: 10.14778/3461535.3461552.

Wang, Y., C. Xiao, J. Qin, R. Mao, M. Onizuka, W. Wang, R. Zhang, and Y. Ishikawa. (2021b). "Consistent and Flexible Selectivity Estimation for High-Dimensional Data". In: *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. Ed. by G. Li, Z. Li, S. Idreos, and D. Srivastava. ACM. 2319–2327. DOI: 10.1145/3448016.3452772.

Wang, Y. R., M. Willsey, and D. Suciu. (2023). "Free Join: Unifying Worst-Case Optimal and Traditional Joins". *Proc. ACM Manag. Data*. 1(2). DOI: 10.1145/3589295.

Wiener, J. L., H. A. Kuno, and G. Graefe. (2009). "Benchmarking Query Execution Robustness". In: *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*. Ed. by R. O. Nambiar and M. Poess. Vol. 5895. *Lecture Notes in Computer Science*. Springer. 153–166. DOI: 10.1007/978-3-642-10424-4_12.

Winslett, M. (2002). "David DeWitt speaks out". *SIGMOD Rec.* 31(2): 50–62. DOI: 10.1145/565117.565127.

Wolf, F., M. Brendle, N. May, P. R. Willems, K. Sattler, and M. Grossniklaus. (2018a). "Robustness Metrics for Relational Query Execution Plans". *Proc. VLDB Endow.* 11(11): 1360–1372. DOI: 10.14778/3236187.3236191.

Wolf, F., N. May, P. R. Willems, and K. Sattler. (2018b). "On the Calculation of Optimality Ranges for Relational Query Execution Plans". In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. Ed. by G. Das, C. M. Jermaine, and P. A. Bernstein. ACM. 663–675. DOI: 10.1145/3183713.3183742.

Woltmann, L., C. Hartmann, M. Thiele, D. Habich, and W. Lehner. (2019). "Cardinality estimation with local deep learning models". In: *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*. Ed. by R. Bordawekar and O. Shmueli. ACM. 5:1–5:8. DOI: 10.1145/3329859.3329875.

Wu, P. and G. Cong. (2021). "A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation". In: *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. Ed. by G. Li, Z. Li, S. Idreos, and D. Srivastava. ACM. 2009–2022. DOI: 10.1145/3448016.3452830.

Wu, W. (2013). "Predicting Query Execution Time: Are Optimizer Cost Models Really Unusable?" URL: pages.cs.wisc.edu/~wentaowu/slides/ICDE-2013.pdf.

Wu, W., Y. Chi, H. Hacigümüs, and J. F. Naughton. (2013a). "Towards Predicting Query Execution Time for Concurrent and Dynamic Database Workloads". *Proc. VLDB Endow.* 6(10): 925–936. DOI: 10.14778/2536206.2536219.

Wu, W., Y. Chi, S. Zhu, J. Tatemura, H. Hacigümüs, and J. F. Naughton. (2013b). "Predicting query execution time: Are optimizer cost models really unusable?" In: *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*. Ed. by C. S. Jensen, C. M. Jermaine, and X. Zhou. IEEE Computer Society. 1081–1092. DOI: 10.1109/ICDE.2013.6544899.

Wu, W., J. F. Naughton, and H. Singh. (2016). "Sampling-Based Query Re-Optimization". In: *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. Ed. by F. Özcan, G. Koutrika, and S. Madden. ACM. 1721–1736. DOI: 10.1145/2882903.2882914.

Wu, W., X. Wu, H. Hacigümüs, and J. F. Naughton. (2014). "Uncertainty Aware Query Execution Time Prediction". *Proc. VLDB Endow.* 7(14): 1857–1868. DOI: 10.14778/2733085.2733092.

Yang, Z., A. Kamsetty, S. Luan, E. Liang, Y. Duan, X. Chen, and I. Stoica. (2020). "NeuroCard: One Cardinality Estimator for All Tables". *Proc. VLDB Endow.* 14(1): 61–73. DOI: 10.14778/3421424.3421432.

Yang, Z., E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. (2019). "Deep Unsupervised Cardinality Estimation". *Proc. VLDB Endow.* 13(3): 279–292. DOI: 10.14778/3368289.3368294.

Yu, X., G. Li, C. Chai, and N. Tang. (2020). "Reinforcement Learning with Tree-LSTM for Join Order Selection". In: *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE. 1297–1308. DOI: 10.1109/ICDE48307.2020.00116.

Zaheer, M., S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola. (2017). "Deep Sets". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett. 3391–3401. URL: proceedings.neurips.cc/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html.

Zhou, X., C. Chai, G. Li, and J. Sun. (2022). "Database Meets Artificial Intelligence: A Survey". *IEEE Trans. Knowl. Data Eng.* 34(3): 1096–1116. DOI: 10.1109/TKDE.2020.2994641.

Zhu, J., N. Potti, S. Saurabh, and J. M. Patel. (2017). "Looking ahead makes query plans robust: making the initial case with in-memory star schema data warehouse workloads". *Proc. VLDB Endow.* 10(8): 889–900. DOI: 10.14778/3090163.3090167.

Zhu, R., Z. Wu, Y. Han, K. Zeng, A. Pfadler, Z. Qian, J. Zhou, and B. Cui. (2021). "FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation". *Proc. VLDB Endow.* 14(9): 1489–1502. DOI: 10.14778/3461535.3461539.