Sub-optimal Join Order Identification with L1-error

YESDAULET IZENOV, University of California, Merced, USA
ASOKE DATTA, University of California, Merced, USA
BRIAN TSAN, University of California, Merced, USA
FLORIN RUSU, University of California, Merced, USA

Q-error – the standard metric for quantifying the error of individual cardinality estimates – has been widely adopted as a surrogate for query plan optimality in recent work on learning-based cardinality estimation. However, the only result connecting Q-error with plan optimality is an upper-bound on the cost of the worst possible query plan computed from a set of cardinality estimates—there is no connection between Q-error and the real plans generated by standard query optimizers. Therefore, in order to identify sub-optimal query plans, we propose a learning-based method having as its main feature a novel measure called L1-error. Similar to Q-error, L1-error requires complete knowledge of true cardinalities and estimates for all the sub-plans of a query plan. Unlike Q-error, which considers the estimates independently, L1-error is defined as a permutation distance between true cardinalities and estimates for all the sub-plans having the same number of joins. Moreover, L1-error takes into account errors relative to the magnitude of their cardinalities and gives larger weight to small multi-way joins. Our experimental results confirm that, when L1-error is integrated into a standard decision tree classifier, it leads to the accurate identification of sub-optimal plans across four different benchmarks. This accuracy can be further improved by combining L1-error with Q-error into a composite feature that can be computed without overhead from the same data.

CCS Concepts: • **Information systems** → **Query optimization**; **Query planning**.

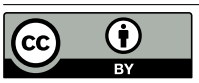
Additional Key Words and Phrases: join ordering, cardinality estimation, permutation distance, database query processing, feature engineering, "bad" query plans

ACM Reference Format:
Yesdaulet Izenov, Asoke Datta, Brian Tsan, and Florin Rusu. 2024. Sub-optimal Join Order Identification with L1-error. *Proc. ACM Manag. Data* 2, 1 (SIGMOD), Article 17 (February 2024), 24 pages. https://doi.org/10.1145/3639272

1 INTRODUCTION

Q-error is the standard metric for quantifying the error of individual cardinality estimates [33]. It is defined formally as the maximum quotient between the estimated and true cardinalities, thereby equally penalizing both overestimations and underestimations. Q-error is widely adopted in recent work on learning-based cardinality estimation methods [4, 8, 19, 21, 44] as a surrogate for the quality of query execution plans, which is measured by P-error—the ratio between the cost of the selected and optimal query plans [11]. However, the only theoretical result connecting the two is a worst-case upper-bound [33] stating that the cost of a query plan computed with estimates having a maximum Q-error of Q is at most Q^4 times larger than the cost of the plan computed with true cardinalities—which is assumed to be optimal. Given only estimated and true cardinalities, the

Authors' addresses: Yesdaulet Izenov, University of California, Merced, Merced, California, USA, yizenov@ucmerced.edu; Asoke Datta, University of California, Merced, Merced, California, USA, adatta2@ucmerced.edu; Brian Tsan, University of California, Merced, Merced, California, USA, btsan@ucmerced.edu; Florin Rusu, University of California, Merced, Merced, California, USA, frusu@ucmerced.edu.



ACM 2836-6573/2024/2-ART17
https://doi.org/10.1145/3639272

bound provides a rough idea of how bad the worst possible query plan could be. Then, it is obvious that for small values of the Q-error, the gap between the worst and optimal plan is relatively small.

Since real query optimizers aim to identify the optimal plan – not the worst – we investigate how useful is the Q-error in assessing the optimality of a query plan based solely on cardinality estimates. For this, we compute the optimal plan using true cardinalities while the database plan is derived using PostgreSQL estimates. These two sets of values are fed into an exhaustive plan enumeration algorithm over a search space consisting of plans with arbitrary structure—including left-deep and bushy. The cost of a plan is computed using the cost function defined in Eq. (1), which is introduced in [25]. This cost function considers both hash and index nested loop joins.

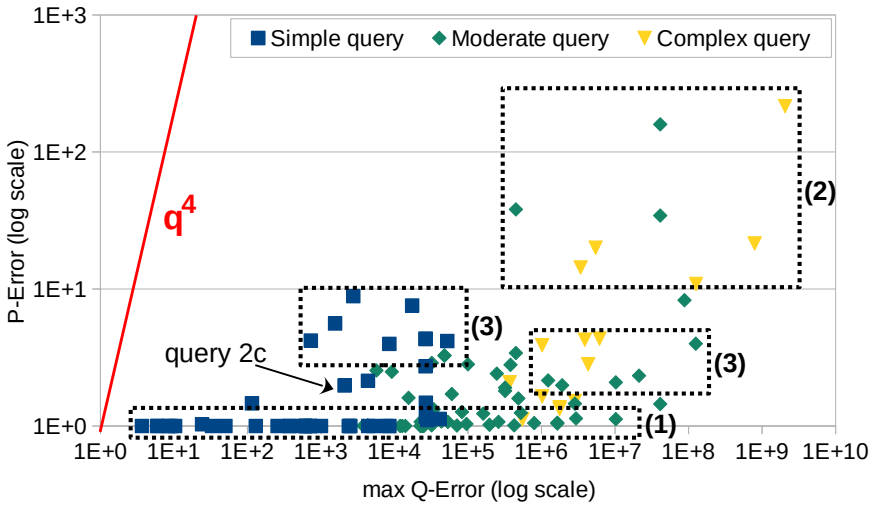


Fig. 1. P-error and Q-error are computed for all 113 JOB queries. Queries are grouped by complexity: 45 Simple with 4-9 joins, 53 Moderate with 10-19 joins, and 15 Complex with 20-28 joins.

As shown in Figure 1, which displays P-error as a function of the Q-error for the queries in the JOB benchmark [26], there is no observable relationship between the two beyond the worst-case upper bound. First, a large number of simple and moderate queries have optimal plans – P-error close to 1 – even though they exhibit large variation on the Q-error—more than six orders of magnitude **(1)**. No matter if the Q-error is 10 or 10^7, an optimal plan can be selected using the same cardinality estimates. Second, we intuitively expect that Q-error is somewhat correlated with P-error—as the Q-error increases, so does the P-error. This happens only for a limited number of complex queries **(2)**. These cases imply the selection of sub-optimal plans with different join orders. Third, the results include queries for which the relationship between Q-error and P-error is reversed **(3)**. The P-error of a query is larger than the P-error of another query even though its Q-error is smaller. This type of inversion shows that the relationship between the two errors is not even monotonic. Finally, since the plans selected by the optimizer are far away from the worst-case, we argue that Q-error falls short as an indicator for the sub-optimality of query plans. Therefore, we need to consider alternative metrics that focus on the real query plans generated by an optimizer rather than the worst-case plan.

Problem. Given a query plan generated by an optimizer based on a set of cardinality estimates, our goal is to determine if it is sub-optimal. Even though we consider general cost functions that include physical operator details, our understanding of sub-optimality is mostly with respect to the

join order. We define sub-optimality in terms of P-error: a plan with a cost at least c times larger than the cost of the plan computed with true cardinalities, where c is a user-defined parameter, is considered sub-optimal. Therefore, we make the implicit assumption that true cardinalities generate the optimal plan. Identical to Q-error computation, sub-optimal plans are identified outside of the runtime query optimization process and they require complete knowledge of true cardinalities and estimates.

High-level approach. We propose a learning-based method for the sub-optimal plan identification problem. We treat identification as binary classification, where plans with a cost at least c times larger than the optimal are considered sub-optimal—the other plans are optimal. The focus of our work is on finding the best features for the classifier—not on designing a new classifier. To this end, we employ standard decision trees. The classifier is trained on a workload of query plans correctly labeled and is expected to accurately predict sub-optimal plans from a testing dataset.

L1-error. The main contribution of this work is the design of the L1-error feature for sub-optimal plan classification. Similar to Q-error, L1-error requires complete knowledge of true cardinalities and estimates for all the sub-plans of a query plan. Unlike Q-error, which considers the estimates independently, L1-error is defined as the distance between the permutations – orders – corresponding to true cardinalities and estimates for all the sub-plans having the same size, i.e., the number of joins. Intuitively, the more different the two permutations are, the higher the chance the plan computed using estimates is significantly different than the optimal plan, thus, likely sub-optimal. The same observation is made in [33], where a plan is known to be optimal if the two permutations are identical. We move beyond this limited case and define a quantitative measure for the difference between permutations. Moreover, L1-error takes into account errors relative to the magnitude of cardinalities since larger cardinalities have a greater influence on plan optimality and, hence, their errors should incur higher penalties. L1-error also considers that small multi-way joins are more critical, with their cardinality estimates likely to be more accurate than larger joins [26, 38].

We summarize our main technical contributions as follows:

- We conduct an in-depth data-driven analysis of the impact of Q-error on cardinality estimation and query plan optimality (Section 4). We study how Q-errors are distributed across different join sizes and their impact on finding optimal query plans. Moreover, we identify practical limitations of Q-error as a feature for sub-optimal plan classification.
- We introduce the L1-error feature for identifying sub-optimal query plans (Section 5). L1-error is specifically tailored to assess how cardinality estimation errors impact plan enumeration algorithms. It is designed to bridge the cardinality estimation errors and enumeration algorithms, ultimately enhancing the interpretability of query optimizer performance.
- We apply L1-error as the single feature of a standard decision tree classifier (Section 6) and evaluate its accuracy in identifying sub-optimal query execution plans over four different benchmarks, including JOB [25, 26], JOB-light [21], JCCH [2], and DSB [7] benchmarks (Section 7). Our experimental results confirm that L1-error is an accurate feature for identifying sub-optimal plans. This accuracy can be further improved by combining L1-error with Q-error into a composite feature that can be computed without overhead from the same data.

2 RELATED WORK

Cardinality estimation errors. The importance of each component within a query optimizer is widely acknowledged in comprehensive studies on query optimization [5, 29]. However, Leis et al. empirically prove that cardinality estimates hold paramount importance [25, 26]. They observe instances where cardinality miscalculations do not inevitably lead to sub-optimal query plans. This is because, provided the errors in misestimation are evenly distributed across a large portion of the

```
SELECT COUNT(*)
FROM company_name cn, keyword k, movie_keyword mk, movie_companies mc, title t
WHERE k.keyword = 'character-name-in-title' and cn.country_code = '[sm]'
  and mc.movie_id = mk.movie_id and cn.id = mc.company_id and k.id = mk.keyword_id and t.id = mk.movie_id and t.id = mc.movie_id
```

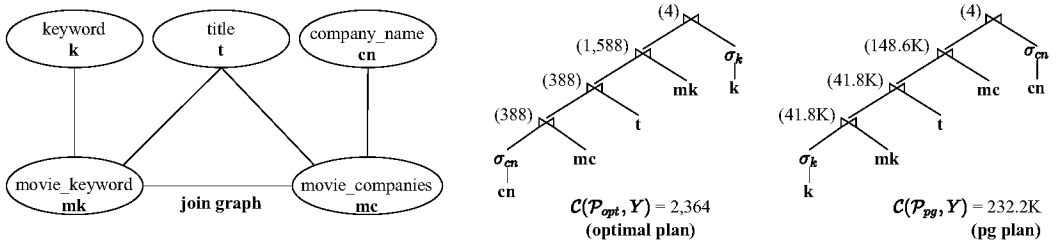


Fig. 2. SQL statement for JOB query 2c, its corresponding join graph, and the query plans selected using true cardinalities Y (for the optimal plan $\mathcal{P}_{opt}$) and cardinality estimations $\hat{Y}$ (for the PostgreSQL plan $\mathcal{P}_{pg}$). The join sizes shown in parentheses are exact – not estimates – while the plan costs $C(\mathcal{P}_{opt}, Y)$ and $C(\mathcal{P}_{pg}, Y)$ are also computed using true cardinalities.

estimates, sub-optimal query plans can be avoided. Consequently, estimation consistency can be more important than high accuracy, since consistency does not disrupt the ranking of query plans. Perron et al. [38] also confirm the importance of cardinality estimations as well as the importance of cardinality estimation of smaller joins than higher-level joins. Based on the JOB benchmark, their findings reveal that obtaining the true cardinalities up to four-way joins suffices to achieve near-optimal runtime performance. This highlights the need for accurate and consistent cardinality estimations, particularly for lower-level joins, in optimizing query performance.

Indicators of sub-optimal plans. Moerkotte et al. [33] introduce Q-error as a measure of individual join cardinality misestimation. At the query level, the authors propose leveraging the maximum Q-error across all sub-query cardinality estimates as a theoretical upper-bound to identify sub-optimal query plans. This serves as the upper-bound for P-error, which is the cost ratio between the selected and optimal plans, to denote the optimality of query plans. However, Han et al. highlight the limitations of the Q-error bound as an indicator of query plan optimality [11]. They observe that Q-error treats all cardinalities with equal weights, which is not always reflective of the realities of query optimizers. As a result, even in the presence of significantly large Q-error values, a query optimizer can still choose an optimal plan, and, conversely, a low Q-error is not always an indicator of an optimal plan.

3 QUERY OPTIMIZATION

The goal of query optimization [5, 25, 26, 29] is to find the optimal query execution plan – the plan with the fastest runtime – for a given SQL query. In classical cost-based query optimization, this is achieved through a search procedure that consists of three components: cost model, cardinality estimation, and plan enumeration. In this section, we provide a brief overview of these components based on the JOB benchmark query 2c, which is depicted in Figure 2. This query joins five tables with five join predicates – including a triangle cycle among tables *t*, *mk*, and *mc* – and has two point selection predicates σ on tables *cn* and *k*. The figure also includes the join graph $G(V, E)$, in which every table is represented as a vertex v connected by edges e for the corresponding join predicates. For example, the join predicate *cn.id = mc.company_id* is represented as the edge between the *cn* and *mc* vertices of the join graph.

A query plan $\mathcal{P}$ defines the sequence in which the tables – vertices – are to be joined. It also dictates the physical operators, such as join or scan, that are deployed during query execution. A query sub-plan can be defined as a query plan that operates over a subgraph of the original join graph. Two plans $\mathcal{P}_{opt}$ and $\mathcal{P}_{pg}$ derived from the join graph are displayed in Figure 2. These plans provide the sequence of tables to be joined — $(cn \bowtie mc \bowtie t \bowtie mk \bowtie k)$ and $(k \bowtie mk \bowtie t \bowtie mc \bowtie cn)$, respectively. Sub-plan $\mathcal{P}_{cn \bowtie mc}$ derived from a subgraph of two vertices *cn* and *mc* of the join graph determines the first two tables to be joined in plan $\mathcal{P}_{opt}$. Throughout the paper, we use the terms *query plan* and *join order* interchangeably.

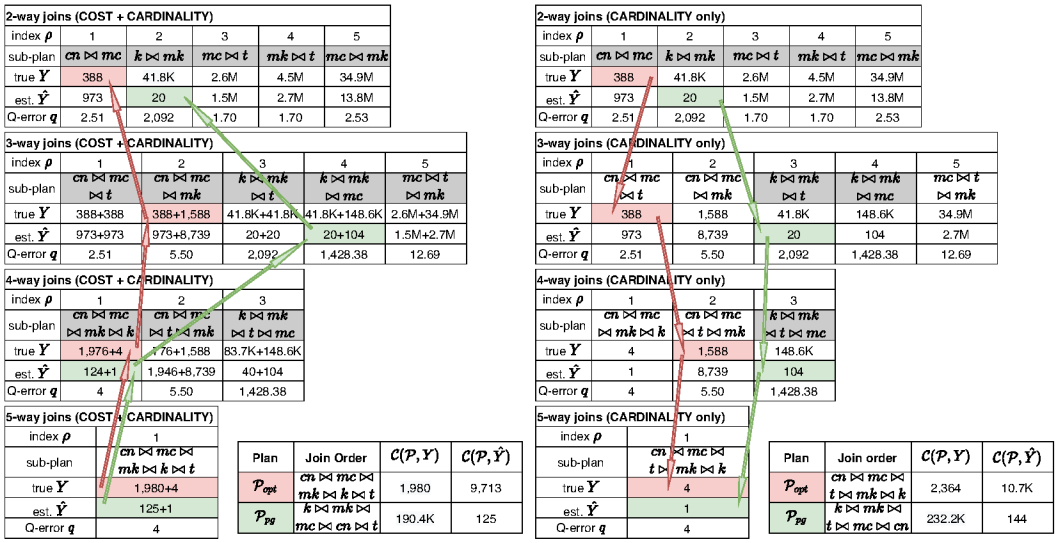
3.1 Cost Model

To evaluate and compare the efficiency of various query plans derived from the join graph, the query optimizer assigns a cost to each of them using an analytical cost function. The query plan considered to be optimal is the one with the minimum cost, as it is expected to have the fastest execution time. However, defining a cost function that can accurately reflect the execution time is challenging. For main-memory databases – the setting for this work – several cost functions have been proposed in past studies [10, 11, 26, 34, 45]. Virtually all these cost functions are defined in terms of the number of tuples – or cardinality – processed by the operators in the query plan. In this work, we settle for the cost function introduced by Leis et al. [25] – which is formally defined in Eq. (1) – because it can accurately predict the query runtime:

$$C(\mathcal{P}) = \begin{cases} \tau \times |R| & \textbf{if } \mathcal{P} = R \vee \mathcal{P} = \sigma(R) \\ |\mathcal{P}| + |\mathcal{P}_1| + C(\mathcal{P}_1) + C(\mathcal{P}_2) & \textbf{if } \mathcal{P} = \mathcal{P}_1 \bowtie^{HJ} \mathcal{P}_2 \\ C(\mathcal{P}_1) + \lambda \times |\mathcal{P}_1| \times \max\left(\frac{|\mathcal{P}_1 \bowtie R|}{|\mathcal{P}_1|}, 1\right) & \textbf{if } \mathcal{P} = \mathcal{P}_1 \bowtie^{INL} \mathcal{P}_2 \ \wedge (\mathcal{P}_2 = R \vee \mathcal{P}_2 = \sigma(R)) \end{cases} \tag{1}$$

This cost function recursively sums the cost of all the nodes in the query plan starting from the leaves – corresponding to scan operators – and following through the joins. In the leaf nodes, the size of a base table R is multiplied by a parameter $\tau = 0.2$ to differentiate between a sequential and an indexed scan. For the intermediate join nodes, the cost function considers two different implementations—hash join $\bowtie^{HJ}$ and index nested-loop join $\bowtie^{INL}$. The hash table is built on the child with the smaller cost—as in zig-zag trees. To differentiate between hash lookup and index lookup, parameter $\lambda = 2$ is used under the assumption that indexes are available on all the join attributes. Even though the cost function considers different physical operators, operator cardinality remains the main factor. This is in line with standard disk-based cost functions, which replace tuple cardinality with block cardinality.

The order in which the tables are joined is pivotal as it can significantly influence the execution time. A sub-optimal ordering can result in unnecessary computational work or data movement, leading to time and resource inefficiencies. The costs of two join orders $\mathcal{P}_{opt}$ and $\mathcal{P}_{pg}$ are displayed in Figure 2. These costs are obtained by summing up the exact cardinalities Y of the three intermediate and the final join. We do not consider the physical operators in order to simplify the presentation. In the case of the optimal plan $\mathcal{P}_{opt}$, the cost $C_{out}(\mathcal{P}_{opt}, Y) = 388 + 388 + 1,588 = 2,364$ is the sum of the cardinalities of sub-plans corresponding to the 2-way join $\mathcal{P}_{cn \bowtie mc}$, the 3-way join $\mathcal{P}_{cn \bowtie mc \bowtie t}$, and the 4-way join $\mathcal{P}_{cn \bowtie mc \bowtie t \bowtie mk}$, respectively. It is important to notice that the cardinality of $\mathcal{P}_{cn \bowtie mc \bowtie t}$ is not the sum of the cardinality for $\mathcal{P}_{cn \bowtie mc}$ and $\mathcal{P}_{mc \bowtie t}$ – edges $(cn - mc)$ and $(mc - t)$ from the join graph – which are 388 and $2.6M$, respectively. This is because only the tuples in the 2-way join $cn \bowtie mc$ are subsequently joined with t—not all the tuples in mc.



(a) Exhaustive plan enumeration (b) Greedy plan enumeration

Fig. 3. Query plan enumeration over the search space corresponding to JOB query 2c.

3.2 Cardinality Estimation

The cost model is defined based on the cardinality of the join operators. Computing the exact cardinality Y of a join is only possible by executing the join operation. This poses a contradiction to the goal of query optimization, which is intended to ascertain the most efficient order for executing the joins without actually performing them. Cardinality estimation steps in to resolve this issue. The function of cardinality estimation is to predict the join cardinality without executing the join operation itself. These estimations $\hat{Y}$ are fed into the cost function in lieu of true cardinalities, allowing the query optimizer to compute the cost of the plan. Subsequently, different plans are ranked based on their estimated cost $C(\mathcal{P}, \hat{Y})$ as opposed to their true cost $C(\mathcal{P}, Y)$. As long as the ranking of the query plans remains consistent across true and estimated costs, these estimations can effectively serve as direct a stand-in for the true cardinalities. Looking through a statistical lens, the purpose of cardinality estimation is to maximize accuracy. However, from the vantage point of the cost model, such precision is not necessary. Instead, what matters is that the costs of two distinct query plans maintain the same order when assessed based on both estimates and exact cardinalities. It is worth noting that while accurate estimates naturally imply correct ranking, they are not essential—as long as their errors are comparable. For instance, despite estimates that are a hundred times greater than the actual values, the cost model treats them as equivalent, producing costs that maintain the same order. Consequently, these estimations do not adversely impact the cost model.

3.3 Plan Enumeration

Plan enumeration is the process of generating and evaluating various query plans that are semantically equivalent but with different costs. The challenge is finding an optimal plan, which entails the minimum cost, by exhaustively enumerating a massive number of candidate query plans. These candidate query plans form what is referred to as the search space. The size of the entire search space is determined by the number of binary trees that can be constructed from $|V|$ vertices in

the join graph [26]. Given the factorial size of the search space, finding an optimal query plan is NP-hard [13].

Search space. Traversing the join graph by following the existing edges $e \in E$, it is possible to avoid enumerating binary trees that incorporate cross-joins, which join tables along non-existent edges $e \notin E$. Nonetheless, even the reduced search space still presents a computational challenge due to its large size. A common strategy is to further downsize the search space by restricting the shape of the considered query plans [30, 32]. This approach entails enumerating only those binary trees with a certain shape. Bushy trees and left-deep trees are two common shapes of binary trees. Bushy trees are characterized by an internal node that joins two sub-trees. On the other hand, left-deep trees are identified by their structure where the right child at any given node is always a leaf node, meaning it does not have any child nodes of its own. Two left-deep trees are depicted in Figure 2. Focusing on specific types of binary trees helps to streamline the enumeration process and manage the complexity of the search space, thereby simplifying the discovery of an optimal query plan. However, this reduction in search space may inadvertently omit optimal query plans. This balances the computational feasibility against the query plan optimality. Therefore, the objective is to find a query plan within the reduced search space that is close to the globally optimal plan.

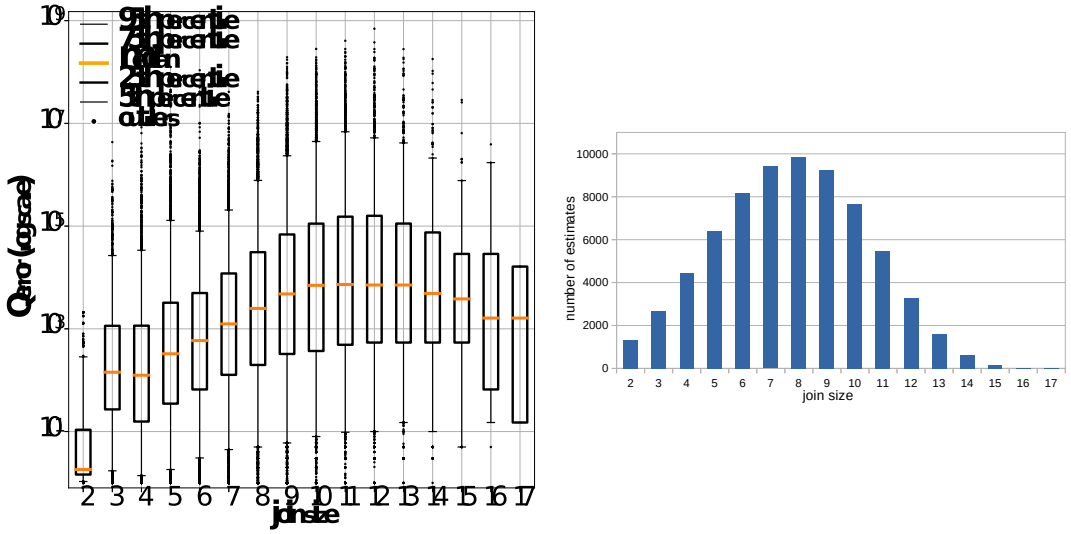
Exhaustive enumeration. To exhaustively enumerate all the trees, each individual query plan is produced and its associated cost is calculated. Figure 3a depicts the optimal plan $\mathcal{P}_{opt}$ – in red – and the PostgreSQL plan $\mathcal{P}_{pg}$ – in green – selected through exhaustive enumeration. $\mathcal{P}_{opt} = (cn \bowtie mc \bowtie mk \bowtie k \bowtie t)$ is computed based on the true cardinalities Y while $\mathcal{P}_{pg} = (k \bowtie mk \bowtie mc \bowtie cn \bowtie t)$ is computed based on the PostgreSQL estimated cardinalities $\hat{Y}$. These plans are shown in the bottom-right table along with their true and estimated costs. As the complexity of a query increases, exhaustive enumeration faces significant computational challenges and is mitigated through optimizations such as dynamic programming and cost-based pruning [6, 31, 40]. Although these optimization techniques can reduce the number of plans evaluated, they still ensure the discovery of the plan with minimum cost—according to the input estimates.

Greedy enumeration. The exhaustive enumeration over the large search space can be further simplified into greedy heuristics that directly compute a single plan [3, 9, 22, 36, 42, 43]. In this case, the cost of a join is its standalone cardinality. Even though the decision at every step is locally optimal, there is no guarantee that the final plan has minimum cost among all the alternative plans. This is due to conditioning the available choices at a step on previous decisions. Figure 3b exhibits this issue for the optimal plan $\mathcal{P}_{opt}$ and the PostgreSQL plan $\mathcal{P}_{pg}$. For both plans, since the plan is built bottom-up starting from 2-way joins, the minimum cardinality 4-way join $\mathcal{P}_{cn \bowtie mc \bowtie mk \bowtie k}$ is not an option because the optimal 3-way joins include t. At the same time, considering fewer sub-plans when building a plan bottom-up means relying on estimates of smaller join size. The estimates for smaller joins are – in principle – more accurate [26]. Thus, while exhaustive enumeration requires consistent estimation across all join sizes, the greedy heuristics are more sensitive to estimates for smaller joins. Therefore, the reduction in the size of the search space can be compensated by consistent estimation of smaller joins.

4 Q-ERROR

Query optimizers often fail to find an optimal plan because of errors and sub-optimal decisions made during the stages of cardinality estimation, cost function, and plan enumeration. In the cost function, inaccuracies in measuring the exact cost of each physical operator in the plan can lead the query optimizer to select a sub-optimal plan. Similarly, attempts to counter computational constraints in plan enumeration through heuristic enumeration and pruning the search space can mislead the query optimizer by excluding optimal plans from its scope of consideration. However,

even under ideal conditions for the cost function and plan enumeration, unavoidable errors in cardinality estimations can jeopardize these stages [25, 26]. In this section, we delve into the widely used Q-error metric [33] that quantifies the errors in cardinality estimation. We discuss how this metric can be employed to evaluate the sub-optimality of a plan by providing better understanding of the extent of estimation errors and their impact on selecting the optimal plan.



(a) Q-error computed using PostgreSQL estimates (b) Number of estimates in exhaustive enumeration

Fig. 4. The distribution of the Q-error and the number of sub-plans as a function of the number of joins for all the 70,407 sub-plans generated from the 113 JOB queries.

4.1 Q-error for Cardinality Estimation

The Q-error metric was proposed as a means to quantify the degree of error in individual cardinality estimations [33]. It has since become a preferred choice for quantifying the accuracy of synopses [4, 38] and learning-based models [19, 21]. Furthermore, it is extensively employed in empirical studies to understand and improve the accuracy of these estimations, thus playing a significant role in query optimization [24, 26, 44]. The Q-error of an individual cardinality estimation is defined as:

$$q_i = \max\left(\frac{\hat{Y}_i}{Y_i}, \frac{Y_i}{\hat{Y}_i}\right) \tag{2}$$

where Y_i and $\hat{Y}_i$ are true and estimated cardinality of a single sub-plan. The Q-error value is in the range of $[1, +\infty)$. In the case of zero values in the denominator, the zeroes can be replaced by a small number—in this work, we use 10^{-4} for this purpose. The Q-error quantifies the deviation of the estimated cardinality $\hat{Y}_i$ from the true cardinality Y_i treating under- and over-estimation equally. In the last row of every table from Figure 3, we show the individual Q-error for the corresponding sub-plan. Q-errors for sub-plans in both exhaustive and greedy plan enumeration are the same. This is because Q-error solely measures the error between true and estimated join cardinalities and does not take into account the sub-plan costs. In the figure, across all join sizes, the Q-error of $\mathcal{P}_{mc\bowtie t}$ and $\mathcal{P}_{mk\bowtie t}$ are the smallest – both equal to 1.70 – which are underestimates of the true cardinalities in this case. The least accurate estimations are for the 2-way join $\mathcal{P}_{k\bowtie mk}$ and the 3-way

join $\mathcal{P}_{k\bowtie mk\bowtie t}$. These Q-errors are equal to 2,092, which also underestimates the true cardinalities. Alternatively, the cardinality of $\mathcal{P}_{cn\bowtie mc\bowtie t}$ is overestimated when the Q-error is 2.51.

In Figure 4a, at each join level, we depict the Q-error measured for the 70,407 sub-plans generated from all 113 queries in the JOB benchmark—excluding cross-joins [25, 26]. The sub-plans are grouped by the number of joins – ranging from 2 to 17 – shown on the x-axis. At each join size, Q-errors are shown via boxplots including 95, 75, 50, 25, and 5 percentiles. Additionally, Figure 4b provides a breakdown of the number of estimates grouped by join size, which illustrates the effect of estimation errors. By examining these estimates, we can gain a clearer understanding of how estimation errors are distributed across different join sizes and how these errors can impact the overall performance of finding optimal query plans. The results show that cardinality estimations for 2-way joins based on 1,336 estimations – which are 1.9% of all sub-plans – are the most accurate. Estimation accuracy from 3-way to 6-way joins starts decreasing—median Q-errors are between 10^1 and 10^3 based on 21,690 estimations, which are 30.8% of all the sub-plans. Starting from 7-way to 13-way joins, median Q-error significantly increases – over 10^3 – which includes 46,544 estimations, or 66.1% of all sub-plans. The rest of the queries form 1.2% of all sub-plans, which is 837 queries. Interestingly, although the number of sub-plans is small, we observe relatively smaller errors starting from 14-way to 17-way joins. These observations indicate that errors increase exponentially with the increase in join size [14]. Inaccuracies in cardinality estimation can have a cumulative detrimental effect on finding optimal query plans. Specifically, significant errors in cardinality estimations at higher-level joins can outweigh and misdirect the query optimizer, causing it to essentially select a query plan at random. Such errors reduce the likelihood of finding the optimal join order and selecting efficient physical operators, thus compromising the overall optimization effectiveness.

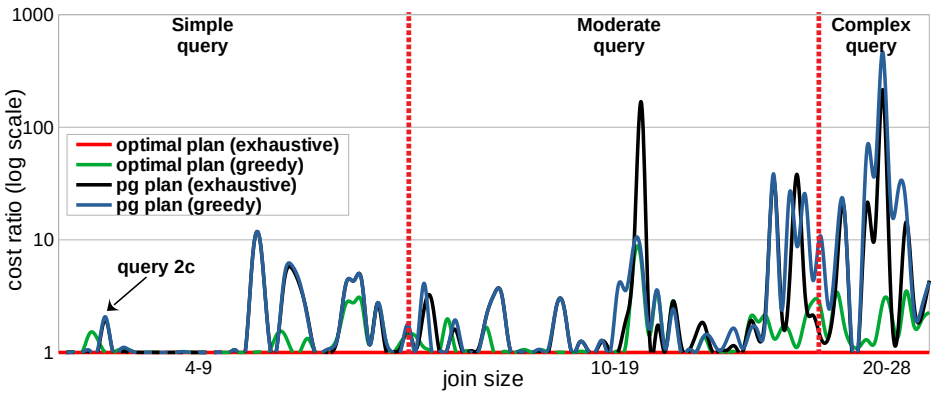


Fig. 5. Impact of cardinality estimation on plan enumeration. The costs are normalized to the cost of the optimal plan obtained by exhaustive enumeration using true cardinalities.

In Figure 5, we illustrate the effect of errors in cardinality estimation on plan enumeration. This figure provides a visual understanding of how inaccuracies in cardinality estimation can influence the performance of different plan enumeration algorithms, highlighting the importance of accurate estimations in finding an optimal query plan. For every JOB query – shown on the x-axis grouped by join size – we compute the cost C of four different plans selected by exhaustive and greedy enumeration when utilizing both true cardinalities Y and the PostgreSQL estimations $\hat{Y}$. These costs are plotted relative to the cost of the exhaustive plan with true cardinalities $C(\mathcal{P}_{opt}, Y)$—the horizontal solid red line at 1. The plans $C(\mathcal{P}_{opt}, Y)$ are optimal within the search space. As expected,

the costs of these optimal plans are lower than the costs of the other plans—equal to or above the red horizontal line. We also observe that several plans selected by the greedy enumeration based on true cardinalities have a higher cost than the greedy plans computed with estimates—green spikes above the blue line. These cases occur when misestimated cardinalities $\hat{Y}$ accidentally lead the greedy enumeration to more efficient plans than those selected based on true cardinalities Y.

The figure also demonstrates the growing need for accurate estimations as the query complexity increases—the gaps among the red, black, and blue lines around the second red vertical line. This behavior is expected as we begin to observe significant misestimations starting from 7-way joins and beyond—as shown in Figure 4a, the Q-error exceeds 10^3 in these cases. Moreover, these significant misestimations adversely impact the exhaustive plan enumeration, as the cumulative effect of estimation errors misguides the cost and enumeration components. For moderate and complex queries, plans selected by exhaustive enumeration have higher costs than plans greedily selected based on estimations—black spikes above the blue line. This means the greedy search algorithm appears to make better decisions based on relatively accurate early-stage join estimations while the exhaustive enumeration is misguided by large misestimations of larger joins. Consequently, the supposedly optimal plans are underestimated compared to the actual optimal plans. Hence, they are selected by the exhaustive enumeration algorithm. These observations suggest that the advantage of exhaustive search diminishes when operating on significant misestimations of large joins. Such errors hinder the enumeration algorithm from finding the optimal query plan.

4.2 Q-error for Plan Optimality

In addition to measuring the error of an individual estimate, Moerkotte et al. [33] introduce a theoretical upper-bound on the ratio of the cost of the selected $\mathcal{P}_{pg}$ and optimal $\mathcal{P}_{opt}$ query plans using Q-error:

$$\frac{C(\mathcal{P}_{pg}, Y)}{C(\mathcal{P}_{opt}, Y)} \leq q^4 \tag{3}$$

where $q = \max_{i \subseteq X} \{q_i\}$ and X is the set of all sub-plans. The cost ratio between the selected and optimal plan has recently been named P-error [11, 27, 34]. In Figure 3, for query 2c, the maximum Q-error is $q = 2,092$. The bound states that, given the estimated and true cardinalities of a query, we can determine whether the selected plan is equivalent to the optimal plan without having to enumerate and select expected and optimal query plans to compute P-error. In other words, if the P-error of the selected and optimal plan is at most q^4, then the selected plan is identical – or close – to the optimal plan. This approach provides a theoretical method to evaluate the optimality of a query plan based on cardinality estimates, reducing the computational burden associated with exhaustive enumeration.

To evaluate the Q-error bound on a larger and more complex workload operating on real-world data, we compute both the Q-error and P-error for the entire JOB benchmark consisting of 113 queries [25, 26]. The results are presented in Figure 1 from the Introduction 1. We categorize the queries based on the number of joins: 45 simple queries with 4-9 join predicates, 53 moderate queries with 10-19 join predicates, and 15 complex queries with 20-28 join predicates, respectively. To compute the P-error for a query, the join orders $\mathcal{P}_{opt}$ and $\mathcal{P}_{pg}$ are determined by the exhaustive enumeration algorithm with both true and PostgreSQL estimated cardinalities. The figure also includes the upper-bound from Eq. (3) as an exponential function—represented by the red line.

The results show a large number of simple and moderate queries with optimal plans – P-error equal to 1 – despite having a high Q-error. The P-error for these queries indeed falls within the bound, hence their plans are optimal. However, many other queries that comply with the bound exhibit significantly larger P-error and even higher Q-error. These queries are primarily moderate

or complex queries having plans that are not optimal, thus, resulting in different join orders despite satisfying the bound. For query 2c from Figure 3a, the optimal plan $\mathcal{P}_{opt}$ is $(cn \bowtie mc \bowtie mk \bowtie k \bowtie t)$, which has the true cost $C(\mathcal{P}_{opt}, Y) = 1,980$. The PostgreSQL selected plan $\mathcal{P}_{pg}$ is $(k \bowtie mk \bowtie mc \bowtie cn \bowtie t)$, which has the true cost $C(\mathcal{P}_{pg}, Y) = 190.4K$. This results in a P-error of 96 and a Q-error of $q = 2,092$. According to the bound [11], $96 \leq 2,092^4 \approx$ 2E+13 is correct. However, the gap between the two values is immense—more than 10 orders of magnitude. For moderate and complex queries, we observe larger cost deviations and extremely high Q-error values. This indicates that the selected and optimal query plans are very different despite satisfying the Q-error bound. Therefore, we argue that Q-error is too loose as a bound and, as an indicator, fails to identify sub-optimal query plans. Intuitively, we expect a small P-error to correspond with a small Q-error and a large P-error with a large Q-error. However, our observations show that queries with a large P-error can have a small Q-error and vice versa. Consequently, these observations show that the maximum Q-error bound falls short in accurately determining the optimality of query plans [11, 35].

5 L1-ERROR

Finding an optimal join order highly depends on the accuracy of cardinality estimations and how the misestimation errors "impact" the plan enumeration algorithm. In Figure 3, we show the sub-plans enumerated by exhaustive and greedy plan search algorithms sorted by the true cardinality Y. The sorted sub-plans are shown for every join size k of query 2c. For 2-way joins, if the sub-plans are sorted by the estimated cardinality $\hat{Y}$, then the relative order becomes different from the order of the sub-plans sorted by true cardinality Y. The difference occurs because of the impact of cardinality misestimation errors of sub-plans $cn \bowtie mc$ and $k \bowtie mk$. However, this difference is minimal in this case – only on one position – henceforth, the plan enumeration algorithm is likely to make accurate decisions in choosing optimal sub-plans. In this section, we introduce the L1-error to quantify the permutation distance between the order of the true cardinalities and that of the estimates for a given sub-plan size. While the importance of the relative ordering of the estimates has been pointed out in previous work [33], L1-error is the first measure to analytically quantify and employ it in determining if a particular join order is optimal.

5.1 Relative Sub-Plan Arrangement

By sorting the sub-plans for each join size k by true and estimated cardinality, we have two sub-plan arrangements — position vectors of the same length. For instance, in 2-way joins, the two position vectors are defined as $\rho = (1, 2, 3, 4, 5)$ sorted by Y and $\widehat{\rho} = (2, 1, 3, 4, 5)$ sorted by $\hat{Y}$, respectively. To differentiate between ρ and $\widehat{\rho}$, we name ρ as the identity permutation in the rest of the paper. Similarly, for each join size k, both position vectors are denoted as ρ and $\widehat{\rho} \in \mathbb{N}^d$ by $(1, 2, \ldots, d)$ where d is the number of sub-plans of size k. There is an extensive range of metrics available, such as Spearman's footrule [41] and Kendall's tau [18], to measure the distance between two arrangements (or ranked lists) [28]. This field has been extensively researched and is a well-studied area. Among the various metrics available, we find Spearman's footrule distance (also known as L1 distance) is particularly effective in quantifying the impact of cardinality estimation errors on plan search algorithms. It provides a strong measure of the distance between two sub-plan arrangements. While it is certainly possible to substitute Spearman's footrule distance with other metrics to compare precision performances, we show, through our observations, that Spearman's footrule distance intuitively fits our problem well and delivers accurate results. In the remainder of this paper, for the sake of convenience, we refer to Spearman's footrule distance as L1-error. L1-error measures the element-wise absolute differences between two position vectors ρ and $\widehat{\rho}$:

$$\mathrm{L1}^k(\rho, \widehat{\rho}) = \sum_{i}^{d} |\rho(i) - \widehat{\rho}(i)| \tag{4}$$

where k is join size, and $\rho(i)$ and $\widehat{\rho}(i)$ are the positions of i-th sub-plan in the position vectors ρ and $\widehat{\rho}$. In the case of identical sub-plan arrangements, $\mathrm{L1}^k(\rho, \widehat{\rho}) = 0$. In the case of discrepancies, L1-error captures the cardinality estimation errors that affect the relative arrangement of sub-plans. In Figure 6, we demonstrate the position vectors ρ and $\widehat{\rho}$ for each join size in query 2c — first two rows of the tables located on the left side of the white vertical bars. The position vectors are generated based on the cardinalities shown in Figure 3b. L1-error measures, $\mathrm{L1}^k(\rho, \widehat{\rho})$, for 2, 3 and 4-way joins are 2, 8 and 2, respectively.

5.2 Weighted Relative Sub-Plan Arrangement

From Equation 4, we observe that the L1-error does not take into account the following criteria which are important to capture for plan search algorithms:

- Significantly over and underestimating cardinality should be associated with greater penalties. Conversely, mispositioning sub-plans with similar cardinalities in the position vectors should carry fewer and relatively similar penalties.
- Cardinality misestimations that occur early in the position vector should attract greater penalties. This is particularly beneficial for plan search algorithms, which are more likely to choose a sub-plan from the first half of the position vector.

Given the limitations identified in the original L1-error, we propose an enhanced L1-error. This new design seeks to improve upon the original by more effectively capturing the impacts of cardinality misestimations on plan search algorithms in query optimization. Kumar et al. [23] propose a method for measuring the distance between two ranked lists, with extensions to factor in the weights of elements and positions, as well as similarities between elements. In our study, we adapt and employ variations of these proposed extensions that align intuitively with our problem context.

2-way join (CARDINALITY only)					
ρ	1	2	3	4	5
$\hat{\rho}$	2	1	3	4	5
Swap cost	1.0	107.84	62.36	1.73	7.71
Monotonic weight	1.0	108.84	171.2	172.93	180.64

3-way join (CARDINALITY only)					
ρ	1	2	3	4	5
$\hat{\rho}$	3	4	1	2	5
Swap cost	1.0	4.09	26.35	3.55	234.69
Monotonic weight	1.0	5.09	31.44	34.99	269.68

4-way join (CARDINALITY only)			
ρ	1	2	3
$\hat{\rho}$	1	3	2
Swap cost	1.0	397.0	93.55
Monotonic weight	1.0	398.0	491.55

Impact weight, W (2-way join)					
	1	2	3	4	5
1	1.0	107.84	6,724.56	11,659.61	89,854.74
2		1.0	62.36	108.12	833.26
3			1.0	1.73	13.36
4				1.0	7.71
5					1.0

Impact weight, W (3-way join)					
	1	2	3	4	5
1	1.0	4.09	107.84	382.87	89,854.74
2		1.0	26.35	93.55	21,954.43
3			1.0	3.55	833.26
4				1.0	234.69
5					1.0

Impact weight, W (4-way join)			
	1	2	3
1	1.0	397.0	37,138.0
2		1.0	93.55
3			1.0

Fig. 6. Sub-plan weights used by the plan enumeration algorithms for JOB query 2c.

Sub-plan impact weights. First, we define a misestimation impact of sub-plans within a weight symmetric matrix W for each join size k. Each i-th row of length d in this matrix represents a sub-plan, and misestimation impact weights are defined as:

$$W^k_{i,j} = \max\left(\frac{Y_j}{Y_i}, \frac{Y_i}{Y_j}\right) \tag{5}$$

where $W_{i,j} \geq 1$, and Y_j and Y_i are true cardinality values. In the case of zero cardinality, denominators can be replaced with a small number. The weight scales are comparable across different query complexities because $W_{i,j}$ only captures the relative differences in cardinality. Thus, the specific scales of cardinality in simple and complex queries, and join sizes are not consequential. In Figure 6, for each join size k, we illustrate the computed weight matrix as a separate table on the right side of the white vertical bars. Row and column indexes in each weight matrix represent sub-plan positions in the identity position vector ρ. The weight matrix signifies the relative differences in cardinality between any two sub-plans of the same join size. A larger weight implies a greater disparity between the two sub-plans in terms of their cardinalities. For example, in Figure 3b, the cardinality of 2-way sub-plan $\mathcal{P}_{cn\bowtie mc}$ is 388 and it is significantly less than the cardinalities of the other four 2-way sub-plans with cardinalities of $41.8K$, $2.6M$, $4.5M$, and $34.9M$. Therefore, in row 1 of the weight matrix for $k = 2$, we observe much larger weights. The same weight matrix also exhibits sub-plan weights of similar cardinality, a scenario frequently encountered when estimating sub-plan cardinalities. Cardinalities of sub-plans $\mathcal{P}_{mc\bowtie t}$ and $\mathcal{P}_{mk\bowtie t}$ of join size 2 are relatively close — $2.6M$ and $4.5M$, respectively. Hence, the impact of mispositioning these two sub-plans has less penalty — weight value is 1.73. We define the L1-error that assigns impact penalties for sub-plan misestimations proportionate to their cardinality magnitudes:

$$\mathrm{L1}^k_W = \sum_{i}^{d} \sum_{\substack{j:\rho(j)<\rho(i) \\ \wedge\, \widehat{\rho}(j)>\widehat{\rho}(i)}} W^k_{i,j} \quad + \sum_{\substack{j:\rho(j)>\rho(i) \\ \wedge\, \widehat{\rho}(j)<\widehat{\rho}(i)}} W^k_{i,j} \tag{6}$$

To measure the difference between ρ and $\widehat{\rho}$ with sub-plan misestimation impacts, the outer term sums weights for each sub-plan i when it is mispositioned with other sub-plans. The left inner term aggregates the weights of sub-plans when their true cardinalities are overestimated more than the estimated cardinality of the sub-plan i in $\widehat{\rho}$, despite those sub-plans having true cardinalities smaller than the true cardinality of sub-plan i in ρ. Similarly, the right inner term sums the weights of sub-plans with underestimated true cardinality that is more than the estimated cardinality of the sub-plan i in $\widehat{\rho}$, despite those sub-plans having larger true cardinalities than the true cardinality of sub-plan i in ρ.

For example, in Figure 6, sub-plan $\mathcal{P}_{k\bowtie mk\bowtie t}$ of join size 3 is located at position 3 in ρ. Its true cardinality of $41.8K$ is underestimated as 20 which is less than the estimated cardinalities 973 and 8,739 for sub-plans $\mathcal{P}_{cn\bowtie mc\bowtie t}$ and $\mathcal{P}_{cn\bowtie mc\bowtie mk}$ which have true cardinalities as 388 and 1,588, respectively. Hence, in the weight matrix W^3, the left inner term sums the penalty weights $W_{3,1} = 107.84$ and $W_{3,2} = 26.35$ of $\mathcal{P}_{k\bowtie mk\bowtie t}$ for being misplaced in $\widehat{\rho}$ to the left of sub-plans $\mathcal{P}_{cn\bowtie mc\bowtie t}$ and $\mathcal{P}_{cn\bowtie mc\bowtie mk}$. Similarly, the right term sums the penalty weights for being misplaced to the right side of sub-plans locations in $\widehat{\rho}$ despite their larger true cardinality and higher locations in ρ. However, sub-plan $\mathcal{P}_{k\bowtie mk\bowtie t}$ is not overestimated than sub-plans $\mathcal{P}_{k\bowtie mk\bowtie mc}$ and $\mathcal{P}_{mc\bowtie t\bowtie mk}$. Sub-plan $\mathcal{P}_{k\bowtie mk\bowtie t}$ has true cardinality of $41.8K$ and estimate as 20, while sub-plans $\mathcal{P}_{k\bowtie mk\bowtie mc}$ and $\mathcal{P}_{mc\bowtie t\bowtie mk}$ have true cardinality of $148.6K$ and $34.9M$, and estimates as 104 and $2.7M$, respectively. Thus, the right term is equal to 0 and the overall misposition penalty weight for sub-plan $\mathcal{P}_{k\bowtie mk\bowtie t}$ is 107.84 + 26.35 + 0 = 134.19. For sub-plans of join size $k = 3$, total misestimation impact is $\mathrm{L1}^3_W = 1,221.22$.

Sub-plan position weights. Intuitively, imposing larger penalties for the misestimation of smaller cardinalities is desirable. This is because plan search algorithms are more likely to select sub-plans with smaller cardinalities. The position vectors represent the locations of sub-plans sorted by their true cardinality, thus sub-plans with smaller cardinalities are expected to be positioned early or left half in ρ. In Figure 3, search algorithms, when guided by true cardinality, tend to select sub-plans from the left half of sub-plan lists on every join step. Consequently, preserving the relative order of sub-plans with smaller cardinalities is generally of significant importance. In other words, we impose a higher penalty for the early position differences between position vectors ρ and $\widehat{\rho}$ than the differences at the tail. To achieve desired position-based penalty weights, we need monotonically increasing weights similar to in [23]. We define the cost of a swap between two adjacent sub-plans i and $i-1$ in the position vectors as the ratio between their true cardinalities $Y_i/Y_{i-1} \geq 1$. In Figure 6, we show the swap costs in the third row named as *Swap cost* in each join table. Then, the monotonically increasing swap weights are defined as:

$$\mu_i = \mu_{i-1} + \frac{Y_i}{Y_{i-1}} \tag{7}$$

where $\mu_1 = 1$, and $\mu_i < \mu_j < \mu_k$ such that $i < j$ and $j < k$. In Figure 6, we show the monotonically increasing weights in the fourth row named as *Monotonic weight* in each join table. This monotonic property offers weights by considering both the distance in position and the closeness in cardinality values of the sub-plans. For example, in Figure 6, sub-plan $\mathcal{P}_{k \bowtie mk \bowtie t}$ of join size 3 is at distance 1 from sub-plan $\mathcal{P}_{k \bowtie mk \bowtie mc}$ while $\mathcal{P}_{cn \bowtie mc \bowtie t}$ is away from $\mathcal{P}_{k \bowtie mk \bowtie mc}$ for 3 positions. Similarly, true cardinality 388 of $\mathcal{P}_{cn \bowtie mc \bowtie t}$ is much less than true cardinality $148.6K$ of $\mathcal{P}_{k \bowtie mk \bowtie mc}$ while sub-plan $\mathcal{P}_{k \bowtie mk \bowtie t}$ has true cardinality of $41.8K$. Hence, the respective monotonic weights for these three sub-plans are 1.0, 31.44, and 34.99. Integrating position-based monotonic weights μ into $\mathrm{L1}^k_W$ assigns greater penalties to the misestimation of smaller true cardinalities that are positioned early in the position vector ρ. By assigning these estimations more weight, L1-error more accurately reflects the significance of their errors, improving its ability to predict sub-optimal query plans:

$$\mathrm{L1}^k = \sum_{i}^{d} \mu_i^{-1} \quad \times \quad \left[\sum_{\substack{j:\rho(j)<\rho(i) \\ \wedge\ \widehat{\rho}(j)>\widehat{\rho}(i)}} W^k_{i,j} \quad + \sum_{\substack{j:\rho(j)>\rho(i) \\ \wedge\ \widehat{\rho}(j)<\widehat{\rho}(i)}} W^k_{i,j} \right] \tag{8}$$

In Figure 6, in 3-way joins, penalty weights assigned to the five sub-plans are 490.71, 23.55, 4.27, 13.62, and 0, respectively. L1-error assigns a higher penalty to sub-plans $\mathcal{P}_{cn \bowtie mc \bowtie t}$ and $\mathcal{P}_{cn \bowtie mc \bowtie mk}$, which are 490.71 and 23.55 respectively, because of their earlier positions in ρ. Despite being placed at the fourth position, sub-plan $\mathcal{P}_{k \bowtie mk \bowtie mc}$ has a higher penalty weight of 13.62 than $\mathcal{P}_{k \bowtie mk \bowtie t}$ with 4.27 penalty weight at the third position. This is because $\mathcal{P}_{k \bowtie mk \bowtie mc}$ has a larger cardinality, making it riskier to misplace. Consequently, overall L1-error for 3-way join is $\mathrm{L1}^3 = 532.15$. This illustration exemplifies how L1-error effectively penalizes misestimations based on their impact on plan search algorithms.

6 L1-ERROR FOR PLAN OPTIMALITY

In this section, we explore the potential of the L1 error as a complement or perhaps even an alternative metric to Q-error to evaluate query plans. In addition, we examine how the L1 error, as an independent feature, can be utilized in classifying sub-optimal query plans, thereby demonstrating its efficacy as a reliable indicator.

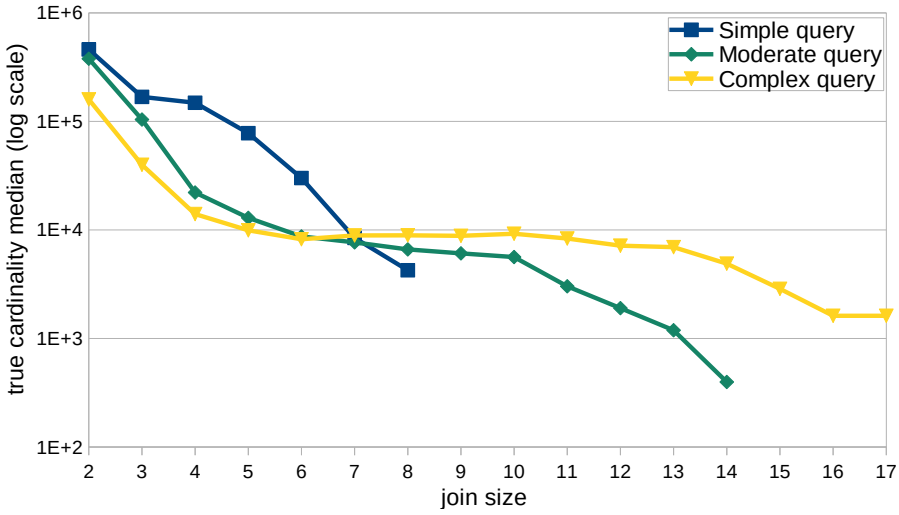


Fig. 7. Importance of small joins at the beginning and decreasing impact of large joins at the end of query plan enumeration algorithms.

6.1 Query-level L1-error

In Section 5, we show how L1^k can independently assess cardinality estimation errors at each join size k. It is essential to note that the number of sub-plans d at each join size k may vary depending on the complexity of the query — different lengths of the position vectors ρ and $\widehat{\rho}$. By aggregating individual L1-error at each join size into L1^k, we can form a feature vector of size K, starting from 2-way joins $k = 2$. Given that queries can have different join sizes, the dimensions of these feature vectors can also vary. We define a query-level L1-error as:

$$\text{L1}_Q = \sum_{k=2}^{K} \text{L1}^k \tag{9}$$

The magnitude of this query-level L1 error can be influenced by the complexity of the query — various numbers of join sizes involved. Consequently, the aggregation approach, which sums up all join-level L1 errors, can be particularly sensitive to the complexity of the query. This implies that more complex queries with a larger number of join sizes may naturally result in higher aggregated L1 errors, emphasizing the influence of query complexity on the overall L1-error calculation. To alleviate this issue, we can take into account our insights from Figure 4a. Cardinality estimations for higher-level joins tend to be severely inaccurate. There is a visible decline in accuracy following 3-way joins, and a considerable drop after 6-way joins. Given this, it is evident that higher-level join cardinality estimations are unreliable and should, therefore, be excluded from consideration or given less weight in comparison to lower-level join cardinality estimations [45]. This approach would counterbalance the tendency for larger errors in complex queries. To further support this decision, we present Figure 7 that illustrates how intermediate data decreases as the join size grows. For each query complexity, we group all JOB sub-plans — including $PK+FK$ and $FK+FK$ joins — by join size and plot the median cardinality value. The results show a consistent trend across all query complexity groups: as join size increases and more filter predicates come into play, intermediate data reduces. Therefore, selecting optimal sub-plans in the early stages of join sizes is crucial to prevent the propagation of large intermediate data. Decisions made at later stages, influenced

by inaccuracies, are less impactful due to the smaller volume of intermediate data processed at higher-level joins. In the figure, we observe a notable drop in data after 6-way join sizes in simple queries, 10-way join in moderate queries, and 15-way join in complex queries. Interestingly, the volume of intermediate data for complex queries between 6 to 13 joins remains stable. This implies the importance of choosing optimal sub-plans up to 13-way joins. While these results are primarily derived from the analysis of JOB sub-plans, we believe this observed trend applies to a wide range of workloads. Taking into account the discussed facts and observed trends, we assign weights to already computed join-level L1-errors L1^k at each join size using an inverse logistic function. For notational purposes, let w_k be defined as:

$$w_k = \frac{e^{-t \times k}}{1 + e^{-t \times k}} \tag{10}$$

where t represents the logistic growth rate or the steepness of the curve. With this in mind, we can define the weighted query-level L1-error as follows:

$$\text{L1}_Q = \sum_{k=2}^{K} w_k \times \text{L1}^k = w^T \times \text{L1} \tag{11}$$

L1-errors across different join sizes are aggregated into a unified L1-error while assigning lower weights to L1-errors at high-level joins. The logistic growth rate t can be tuned based on the performance of the cardinality estimator in use. For example, we set $t = 1.5$, thus $w\_7 = 0.000028$, based on the Q-error values generated by PostgreSQL, depicted in Figure 4a. It begins to reduce the impact of join-level L1-errors starting from 7-way joins as we observe significant estimation errors at higher-level joins.

Algorithm 1 L1-error

Input: largest join size K of input query Q, set of query sub-plans S, true Y and estimated $\hat{Y}$ cardinalities
Output: L1_Q weighted query-level L1-error

function QUERY-L1-ERROR(K, S, Y, $\hat{Y}$)
  $w \in \mathbb{R}^{K-1} \leftarrow$ join size weights from Equation 10
  L1 $\leftarrow$ JOIN-L1-ERROR($K, S, Y, \hat{Y}$)
  return $\text{L1}_Q = w^T \times \text{L1}$
function JOIN-L1-ERROR(K, S, Y, $\hat{Y}$)
  for each join size $k \in \{2 \dots K\}$ **do**
    $S_k \subset S \leftarrow$ subset of sub-plans of size k in S
    $\rho \leftarrow$ positions of S_k increasingly sorted by Y
    $\hat{\rho} \leftarrow$ positions of S_k increasingly sorted by $\hat{Y}$
    // *compute weights from Equations 5 and 7*
    $W^k \leftarrow$ impact weights of sub-plans in S_k
    $\delta^k \leftarrow$ position weights of sub-plans in S_k
    $\text{L1}^k \leftarrow$ L1-error for join size k from Equation 8
  return L1 $\leftarrow$ join-level L1 feature vector of size $K - 1$

Algorithm overview. Algorithm 1 shows an overview of computing L1-error for a given query Q. The inputs are the maximum join size K and all sub-plans S along with their true Y and estimated

$\hat{Y}$ cardinalities. Algorithm 1 consists of two functions Query-L1-error (lines 1-4) and Join-L1-error (lines 5-14). For the sake of clarity, we separate these two functions, although these two functions can be combined. At the query level, in function Query-L1-error, join weights w for each join size k are generated as in Equation 10 (line 2). The impact of joins decreases as their sizes increase following the 'S'-shaped sigmoid curve. Join-level L1 $\in \mathbb{R}^K$ are aggregated into the final query-level L1_Q (line 4). Join-L1-error computes join-level L1^k for each join size k. For the sub-plans S_k of size k, two position vectors ρ and $\hat{\rho}$ are created and sorted in increasing order based on the true Y and estimated $\hat{Y}$ cardinalities, respectively (lines 7-9). Simultaneously, impact weight matrix W^k and position weights δ^k are generated from Equations 5 and 7, respectively (lines 10-12). Lastly, a join-level L1^k for join size k is computed from Equation 8 (line 13). We repeat the process for each join size (lines 6-13) yielding a feature vector, L1 of length K, of join-level L1^k (line 14).

6.2 Applications of L1-error

While minimizing individual Q-errors can enhance the overall efficiency of a query optimizer, the metric often falls short of accurately indicating query plan optimality [11, 44]. Q-error is conventionally used to assess cardinality estimation techniques and synopses, as a separate sub-task that influences the likelihood of finding an optimal query plan [19, 26]. In recent years, Q-error has been widely adopted in learning-based approaches [44], where it is utilized in the post-training evaluation phase [12, 46–48] and during training [21, 34]. In Section 4, we observe that being solely an error measurement, cannot reliably identify sub-optimal query plans. This limitation arises from the fact that other factors that bridge the gap between estimation error and the selection of an optimal plan, such as cost function and plan enumeration, are not taken into account by Q-error.

In this work, we present L1-error as a metric to characterize plan sub-optimality, taking as an input only cardinalities and without requiring plan enumeration to compute P-error. Contrary to Q-error which primarily concentrates on estimation precision, L1-error prioritizes the relative order of sub-plans — a critical aspect for cost function and plan enumeration algorithms. It accounts for the impact weights of sub-plans and their relative displacement in the presence of estimation errors. Therefore, unlike Q-error, L1-error is capable of accurately identifying queries with sub-optimal plans. This suggests that L1-error can serve as a complementary metric to Q-error to evaluate query plans and can be employed in future research to evaluate the sub-optimality of query plans produced by synopses and learning-based models. In the current work, we evaluate L1-error as a standalone measure. For this purpose, we frame the identification of queries with sub-optimal query plans as a binary classification task.

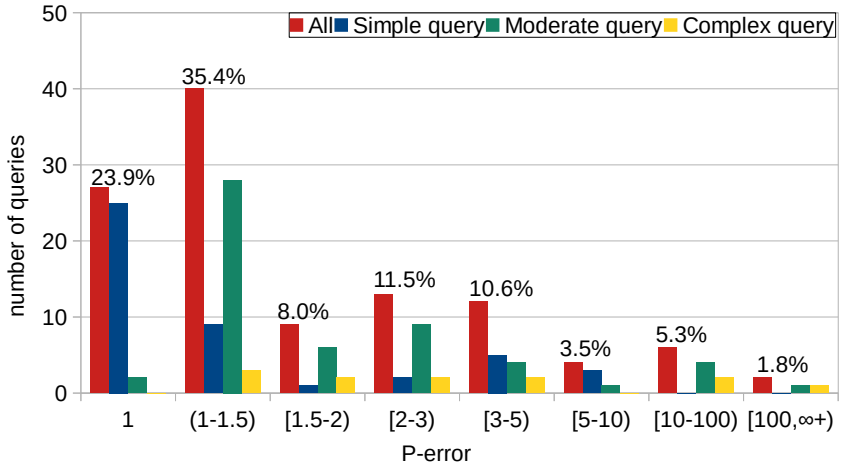
7 EMPIRICAL EVALUATION

In the current study, we assess L1-error as a separate measure and frame the identification of queries with sub-optimal query plans as a binary classification task. This allows us to evaluate the standalone efficacy of L1-error. Our evaluation of L1-error spans three different facets — varying sources of cardinality estimates, plan search algorithms, and workloads and data.

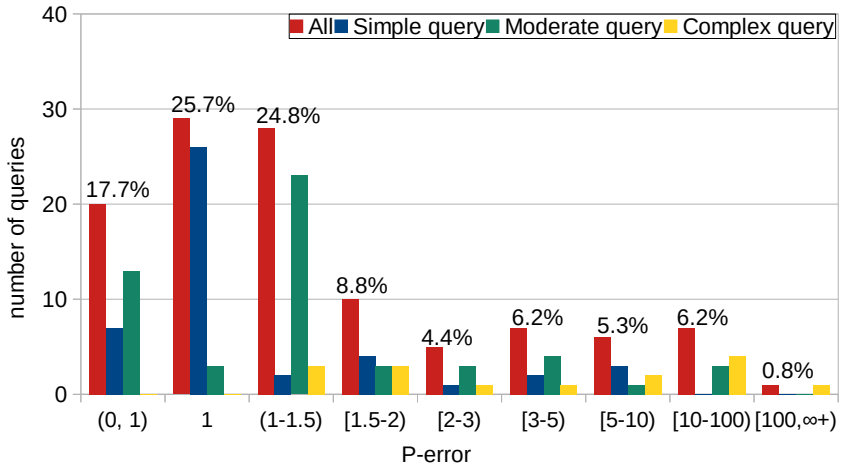
7.1 Experimental Setup

Datasets & query workloads. We perform the experiments on the JOB benchmark [39] over IMDB dataset [1], which has seen extensive use in evaluating query optimizers and has thereby established itself as a standard benchmark [25, 26]. The JOB benchmark defines 113 queries grouped into 33 families. These queries vary significantly in their complexity, with the simplest having 4 join predicates and the largest join size of 4, and the most complex having 28 join predicates with the largest join size of 17. This variability manifests itself in execution times that are highly different. To

compensate for this, we split the queries into three complexity groups — simple (4-9 join predicates), moderate (10-19 join predicates), and complex (20-28 join predicates) — and examine each group separately. We also perform experiments using the JOB-light benchmark [20], a simpler version of the JOB benchmark that includes 67 simple queries that can be represented with a star join graph and feature 2 to 4 join predicates. To show how L1-error generalizes to different workloads and data, we evaluate L1-error on JCCH [2] and DSB [7] benchmarks. We use scale factors of 1 and 10, and generate 511 and 1440 queries, respectively.



(a) Exhaustive enumeration



(b) Greedy enumeration

Fig. 8. Distribution of JOB queries based on P-error using true and PostgreSQL estimated cardinalities.

Methodology & implementation. For cardinality estimations, we select two distinct cardinality estimators—PostgreSQL 15.1 [37], a widely recognized database system, and COMPASS [16, 17], a more recent system. We run the sub-plans of the four workloads in PostgreSQL to collect their estimated and true join cardinalities. Additionally, we collect estimated join cardinalities for the

sub-plans of JOB and JOB-light produced by COMPASS. We compute the P-Error for each query and use it as a true label in our binary classification task. We acquire query plans utilizing exhaustive and greedy plan search algorithms based on estimated and true cardinalities. The cost of the query plans is calculated using the cost function C from Section 3.1.

We show the performance of the L1-error-based binary classifier via confusion matrices depicted in Tables 1 and 2. We label queries with sub-optimal plans as 'positive' (P) and 'negative' (N) otherwise. These classifications are shown in the fifth to eighth columns of the confusion tables. In the tables, we report four measures: 'true positive' (TP), 'true negative' (TN), 'false positive' (FP), and 'false negative' (FN) shown on the ninth to twelfth columns. In addition, we report overall accuracies on test data of the classifiers based on L1-error, Q-error and both in Figures 9 and 10. In the binary classification task, we use a CART decision tree model of a tree depth of 5 from the Scikit-learn library (version 1.1.2). For the logistic growth rate in Equation 10, we set $t = 1.5$, which begins weighting 4-way joins at ≈ 0.002. We partition the queries into training and testing, using a 70% to 30% split, respectively, to have a large enough test data for the classification task. In JCCH and DSB, we split the data into 80% to 20%. The resulting data sizes are shown in the third and fourth columns in Tables 1 and 2. The implementation of the current work together with all the experimental artifacts are available online [15].

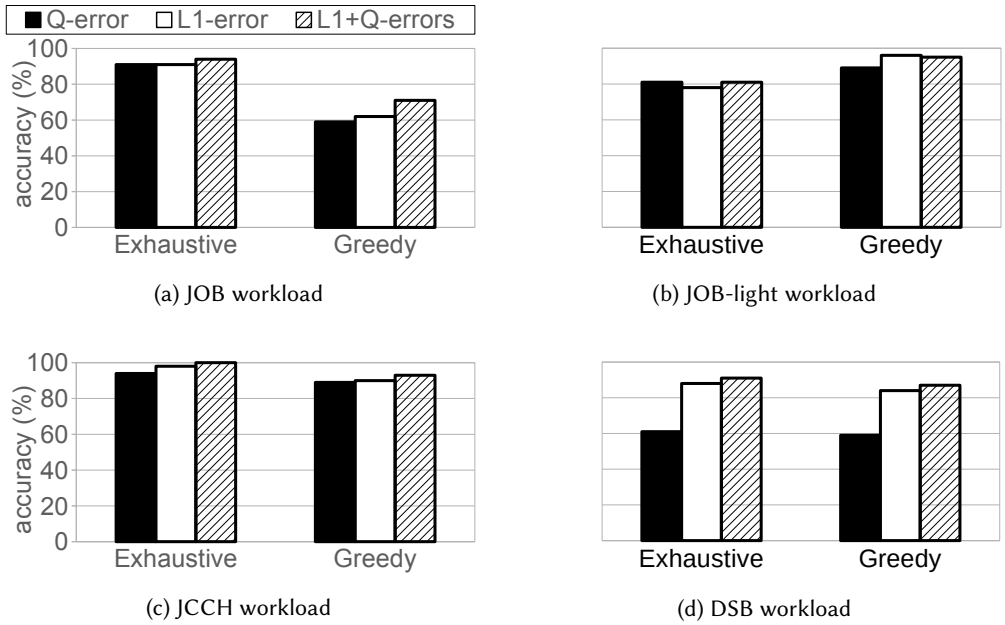
Benchmark	Enumerator	Train queries	Test queries	Actual		Predicted		TP	TN	FP	FN
				Sub-optimal (Positive)	Optimal (Negative)	Sub-optimal (Positive)	Optimal (Negative)				
JOB 113 queries	Exhaustive	79	34	87 61 train *26 test*	26 18 train *8 test*	94 67 train *27 test*	19 12 train *7 test*	85 60 train *25 test*	17 11 train *6 test*	9 7 train *2 test*	2 1 train *1 test*
	Greedy			46 32 train *14 test*	67 47 train *20 test*	51 34 train *17 test*	62 45 train *17 test*	33 24 train *9 test*	49 37 train *12 test*	18 10 train *8 test*	13 8 train *5 test*
JOB-light 67 queries	Exhaustive	40	27	9 5 train *4 test*	58 35 train *23 test*	3 2 train *1 test*	64 39 train *25 test*	1 1 train *0 test*	56 35 train *21 test*	2 0 train *2 test*	8 4 train *4 test*
	Greedy			2 1 train *1 test*	65 39 train *26 test*	0 0 train *0 test*	67 40 train *27 test*	0 0 train *0 test*	65 39 train *26 test*	0 0 train *0 test*	2 1 train *1 test*
JCCH 511 queries	Exhaustive	408	103	150 120 train *30 test*	361 288 train *73 test*	127 99 train *28 test*	384 309 train *75 test*	127 99 train *28 test*	361 288 train *73 test*	0 0 train *0 test*	23 21 train *2 test*
	Greedy			120 96 train *24 test*	391 312 train *79 test*	59 45 train *14 test*	452 363 train *89 test*	59 45 train *14 test*	391 312 train *79 test*	0 0 train *0 test*	61 51 train *10 test*
DSB 1440 queries	Exhaustive	1152	288	727 582 train *145 test*	713 570 train *143 test*	668 529 train *139 test*	772 623 train *149 test*	586 462 train *124 test*	631 503 train *128 test*	82 67 train *15 test*	141 120 train *21 test*
	Greedy			668 534 train *134 test*	772 618 train *154 test*	626 503 train *123 test*	814 649 train *165 test*	529 424 train *105 test*	675 539 train *136 test*	97 79 train *18 test*	139 110 train *29 test*

Table 1. Evaluation of L1-error on query plans selected using PostgreSQL cardinality estimates.

7.2 Results

L1-error performance on PostgreSQL. In this section, we first examine L1-error performance identifying the sub-optimality of query plans selected based on PostgreSQL's cardinality estimation. Subsequently, we analyze L1-error performance on different sets of cardinality estimates collected from the COMPASS estimator used to select query plans. We start the evaluation with the JOB workload, composed of a mix of 113 simple, moderate and complex queries. Out of the four workloads, JOB is the most challenging due to its relatively complex graph topology and large number of joins. Figure 8 displays the distribution of JOB queries, grouped by their P-error shown on the x-axis. In Figure 8a, when employing exhaustive enumeration, we observe that 23.9%

of the selected plans are equivalent to optimal plans (P-error = 1), while 35.4% of 113 queries are near-optimal (P-error < 1.5), thus may be considered successful. The remaining 40.7% of the overall number of queries exhibits larger cost differences, some of which are severely sub-optimal. In the case of greedy enumeration, Figure 8b, we observe that 25.7% and 24.8% of the selected plans are equivalent to optimal plans (P-error = 1) and near-optimal (P-error < 1.5), respectively. Interestingly, 17.7% of the queries have even better plans than the plans selected using true cardinalities (P-error < 1). This is due to greedy decisions made during the enumeration. The remaining 31.8% of the plans demonstrate higher P-errors, and some are severely sub-optimal. These sub-optimal plans mainly include moderate and complex queries, and their sub-optimal plans should be accurately identified.



(a) JOB workload

(b) JOB-light workload

(c) JCCH workload

(d) DSB workload

Fig. 9. L1-error classifier accuracy on test data using PostgreSQL cardinality estimates.

The objective is to directly classify sub-optimal query plans using only L1-error, not by P-error. The first two rows in Table 1 depict the performance of L1-error evaluated on JOB. In the case of exhaustive enumeration, the number of positive queries (sub-optimal) is higher than negative queries (optimal) – columns 5 and 6. Based on the results, we observe a similar trend but now query plans are classified by L1-error as an indicator of queries with sub-optimal plans – columns 7 and 8. The difference between predicted positive and negative queries is also high. Out of 87 true sub-optimal plans (Positive), 85 sub-optimal query plans are correctly classified (TP), resulting in 2 FN. On the other hand, the number of misclassifying optimal plans as sub-optimal (FP) is higher which is not as critical as FN. In the case of greedy enumeration, we observe an opposite trend – the number of positive queries (sub-optimal) is lower than negative queries (optimal). While the classifier results in 18 FP and 13 FN, the predicted Positive and Negative results follow the pattern of the actual Positive and Negative results. Analyzing the misclassified optimal plans, in both enumerations, we observe that the classifier primarily misclassifies queries with P-error < 1.5.

We now evaluate L1-error on simple queries in JOB-light — third and fourth rows in Table 1. Unlike in JOB, this particular workload presents a relatively high unbalanced class ratio for the

classification task. From the results, we notice a significant difference between the number of true sub-optimal plans (Positive of 9 and 2) and true optimal plans (Negative of 58 and 65) in exhaustive and greedy enumerations, respectively. This is expected because, in simple queries, the join sizes are smaller, thus cardinality estimations are relatively accurate. Therefore, this results in a high number of queries with optimal plans in both search algorithms. The results exhibit a similar trend but now queries are classified by L1-error as an indicator. The difference between predicted positive and negative queries is also high. Even though a large P-error may have less impact on the execution time of simple queries, the classifier is still efficient in identifying plan sub-optimality. As in JOB, the misclassified query plans show P-error < 1.5.

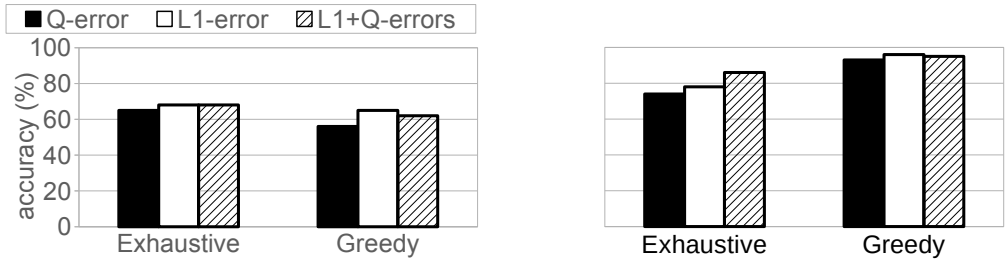
To conduct a comprehensive assessment of L1-error across a more expansive dataset and query spectrum, we present evaluations performed on JCCH (rows 5 and 6) and DSB (rows 7 and 8) workloads. Unlike the previous two workloads, JCCH and DSB show a relatively more balanced class ratio, albeit with the predomination of optimal plans. This is due to the workload complexity standing between JOB and JOB-light. Based on the results, the predicted Positive and Negative results in a similar trend as in actual Positive and Negative results. The predicted class ratio also follows a similar trend. Interestingly, the classifier avoids misclassifying optimal plans as sub-optimal, with 0 FN in both enumeration algorithms. Looking into the misclassified queries reveals query plans exhibiting P-error values centered around 1.78.

Benchmark	Enumerator	Train queries	Test queries	Actual		Predicted		TP	TN	FP	FN
				Sub-optimal (Positive)	Optimal (Negative)	Sub-optimal (Positive)	Optimal (Negative)				
JOB 113 queries	Exhaustive	79	34	78 55 train *23 test*	35 24 train *11 test*	100 68 train *32 test*	13 11 train *2 test*	76 54 train *22 test*	11 10 train *1 test*	24 14 train *10 test*	2 1 train *1 test*
	Greedy			57 40 train *17 test*	56 39 train *17 test*	80 55 train *25 test*	33 24 train *9 test*	55 40 train *15 test*	31 24 train *7 test*	25 15 train *10 test*	2 0 train *2 test*
JOB-light 67 queries	Exhaustive	40	27	14 8 train *6 test*	53 32 train *21 test*	10 4 train *6 test*	57 36 train *21 test*	7 4 train *3 test*	50 32 train *18 test*	3 0 train *3 test*	7 4 train *3 test*
	Greedy			3 2 train *1 test*	64 38 train *26 test*	4 2 train *2 test*	63 38 train *25 test*	3 2 train *1 test*	63 38 train *25 test*	1 0 train *1 test*	0 0 train *0 test*

Table 2. Evaluation of L1-error on query plans selected using COMPASS cardinality estimates.

Figure 9 illustrates the accuracy of the classifier on the test data. We compare the classifier based on L1-error and the classifier based on Q-error as well as the classifier that utilizes both L1-error and Q-error to identify sub-optimal query plans. Overall, we observe an improvement over the Q-error classifier except in exhaustive enumeration on JOB-light attributed to the small number of test data. The results above suggest L1-error is a viable indicator for identifying sub-optimal query plans and can be used in tandem with Q-error to assess query optimizers to identify query sub-optimality. The classifier based on both L1-error and Q-error exhibits overall improvement in identifying sub-optimal plans. The combined approach can provide a more comprehensive evaluation, considering both the absolute accuracy of individual estimates and their impact on query plan optimality.

L1-error performance on COMPASS. In order to evaluate L1-error on a different dimension, we collect true and estimated cardinalities from the COMPASS estimation for JOB and JOB-light workloads. In Table 2, we present the classifier performance identifying sub-optimal query plans selected by exhaustive and greedy enumeration algorithms using COMPASS cardinality estimates. As with PostgreSQL estimates, we observe similar class ratios – actual Positives and Negatives on columns 5 and 6 – in JOB and JOB-light. The predicted Positives and Negatives once again



(a) JOB workload. (b) JOB-light workload.

Fig. 10. L1-error classifier accuracy on test data using COMPASS cardinality estimates.

follow a similar pattern except for greedy enumeration in JOB. The number of estimated Positives is noticeably higher than the actual Positives. Thus, the classifier results in a higher FP while still maintaining a low FN. A closer scrutiny of the misclassified queries reveals a central tendency of query plans exhibiting P-error values around 1.0. In Figure 10, we compare the classifiers using L1-error, Q-error, and both performed, on JOB and JOB-light. As in PostgreSQL estimates, we observe a positive trend over Q-error. The combined classifier on L1-error and Q-error shows improved accuracy on the test data.

7.3 Summary

The experimental results can be summarized as follows:

- L1-error correctly classifies the optimality of query plans by following the trend and ratio between true sub-optimal and optimal query plans. The results contain only a small number of false negatives—the case when true sub-optimal plans are misclassified.
- The classifier based exclusively on L1-error identifies sub-optimal plans more accurately than the classifier that has Q-error as a feature. When having a combined feature consisting of both L1-error and Q-error, the best accuracy is achieved. These results prove that L1-error acts as an important feature both alone as well as in conjunction with Q-error.
- L1-error maintains its accuracy across multiple sets of cardinality estimates and workloads. This proves its generality both for different cardinality estimation synopses as well as across various datasets and queries.

8 CONCLUSIONS AND FUTURE WORK

We introduce L1-error, a novel indicator designed to identify sub-optimal join orders. L1-error emphasizes cardinality estimation errors that influence plan search algorithms, specifically those errors that disrupt the cardinality-based sorted order of sub-plans. Importantly, L1-error disregards estimation errors that do not bear any impact on the plan search algorithms. L1-error also takes into account that the cardinality estimates of earlier multi-way joins tend to be more accurate and critical than those of later joins. Our empirical results, across four different benchmarks, prove that as a standalone metric, L1-error can efficiently identify sub-optimal join orders in both moderate and complex queries.

As we look towards future research, we propose employing L1-error as a supplementary measure in conjunction with Q-error to better correlate with the optimality of a query plan. Therefore, we intend to utilize L1-error in the evaluation of a broader range of cardinality estimation techniques, including learning-based approaches, to assess the efficacy of their trained models.

ACKNOWLEDGMENTS

This work is supported by NSF award number 2008815.

REFERENCES

[1] Peter Boncz. [n. d.]. The IMDB Dataset. http://homepages.cwi.nl/~boncz/job/imdb.tgz.
[2] Peter Boncz, Angelos-Christos Anatiotis, and Steffen Klabe. 2017. JCC-H: Adding Join Crossing Correlations with Skew to TPC-H. In *TPCTC 2017*. 103–119.
[3] Nicolas Bruno, César Galindo-Legaria, and Milind Joshi. 2010. Polynomial Heuristics for Query Optimization. In *ICDE 2010*. 589–600.
[4] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities. In *SIGMOD 2019*. 18–35.
[5] Surajit Chaudhuri. 1998. An Overview of Query Optimization in Relational Systems. In *PODS 1998*. 34–43.
[6] David DeHaan and Frank Tompa. 2007. Optimal Top-Down Join Enumeration. In *SIGMOD 2007*. 785–796.
[7] Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek Narasayya. 2021. DSB: A Decision Support Benchmark for Workload-Driven and Traditional Database Systems. *PVLDB* 14, 13 (2021), 3376–3388.
[8] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates Using Lightweight Models. *PVLDB* 12, 9 (2019), 1044–1057.
[9] Leonidas Fegaras. 1998. A New Heuristic for Optimizing Large Queries. In *DEXA 1998*. 726–735.
[10] Immanuel Haffner and Jens Dittrich. 2023. Efficiently Computing Join Orders with Heuristic Search. *PACMMOD* 1, 1 (2023).
[11] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2022. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *PVLDB* 15, 4 (2022), 752–765.
[12] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *PVLDB* 13, 7 (2020), 992–1005.
[13] Toshihide Ibaraki and Tiko Kameda. 1984. On the Optimal Nesting Order for Computing N-relational Joins. *ACM Transactions on Database Systems* 9, 3 (1984), 482–502.
[14] Yannis E. Ioannidis and Stavros Christodoulakis. 1991. On the Propagation of Errors in the Size of Join Results. *SIGMOD Record* 20, 2 (1991), 268–277.
[15] Yesdaulet Izenov. 2024. Sub-optimal Join Order Identification with L1-error. https://github.com/yizenov/l1-error.
[16] Yesdaulet Izenov, Asoke Datta, Florin Rusu, and Jun Hyung Shin. 2021. COMPASS: Online Sketch-based Query Optimization for In-memory Databases. In *SIGMOD 2021*. 106–117.
[17] Yesdaulet Izenov, Asoke Datta, Florin Rusu, and Jun Hyung Shin. 2021. Online Sketch-based Query Optimization. *CoRR* arXiv:2102.02440v1 (2021).
[18] Maurice G Kendall. 1938. A New Measure of Rank Correlation. *Biometrika* 30, 1/2 (1938), 81–93.
[19] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. 2017. Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models. *PVLDB* 10, 13 (2017), 2085–2096.
[20] Andreas Kipf. [n. d.]. JOB-light Benchmark. https://github.com/andreaskipf/learnedcardinalities/blob/master/workloads/job-light.sql.
[21] Andreas Kipf, Thomas Kipf, Bernhard Radke, Victor Leis, Peter Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR 2019*.
[22] Donald Kossmann and Konrad Stocker. 2000. Iterative Dynamic Programming: A New Class of Query Optimization Algorithms. *TODS* 25, 1 (2000), 43–82.
[23] Ravi Kumar and Sergei Vassilvitskii. 2010. Generalized Distances Between Rankings. In *WWW 2010*. 571–580.
[24] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2021. Survey on Advancing the DBMS Query Optimizer: Cardinality Estimation, Cost Model, and Plan Enumeration. *Data Science and Engineering* 6 (2021), 86–101.
[25] Victor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *PVLDB* 9, 3 (2015), 204–215.
[26] Victor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2018. Query Optimization Through the Looking Glass, and What We Found Running the Join Order Benchmark. *VLDBJ* 27 (2018), 643–668.
[27] Beibin Li, Yao Lu, Chi Wang, and Srikanth Kandula. 2021. Q-error Bounds of Random Uniform Sampling for Cardinality Estimation. *CoRR* arXiv:2108.02715v2 (2021).
[28] Christina Lioma and Niels Dalum Hansen. 2017. A Study of Metrics of Distance and Correlation between Ranked Lists for Compositionality Detection. *Cognitive Systems Research* 44 (2017), 40–49.
[29] Guy Lohman. 2014. Is Query Optimization a Solved Problem? https://wp.sigmod.org/?p=1075.

[30] Guido Moerkotte and Pit Fender. 2013. Counter Strike: Generic Top-Down Join Enumeration for Hypergraphs. *PVLDB* 6, 14 (2013), 1822–1833.
[31] Guido Moerkotte and Thomas Neumann. 2006. Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees Without Cross Products. In *VLDB 2006*. 930–941.
[32] Guido Moerkotte and Thomas Neumann. 2008. Dynamic Programming Strikes Back. In *SIGMOD 2008*. 539–552.
[33] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *PVLDB* 2, 1 (2009), 982–993.
[34] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. *PVLDB* 14, 11 (2021), 2019–2032.
[35] Parimarjan Negi, Ryan Marcus, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2020. Cost-Guided Cardinality Estimation: Focus Where it Matters. In *ICDE Workshops 2020*. 154–157.
[36] Thomas Neumann. 2009. Query Simplification: Graceful Degradation for Join-Order Optimization. In *SIGMOD 2009*. 403–414.
[37] Open Source Relational Database [n. d.]. PostgreSQL. www.postgresql.org.
[38] Matthew Perron, Zeyuan Shang, Tim Kraska, and Michael Stonebraker. 2019. How I Learned to Stop Worrying and Love Re-optimization. In *ICDE 2019*. 1758–1761.
[39] Greg Rahn. [n. d.]. Join Order Benchmark (JOB). https://github.com/gregrahn/join-order-benchmark.
[40] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlain, R. A. Lorie, and T. G. Price. 1979. Access Path Selection in a Relational Database Management System. In *SIGMOD 1979*. 23–34.
[41] Charles Spearman. 1904. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology* 15, 1 (1904), 72–101.
[42] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. 1997. Heuristic and Randomized Optimization for the Join Ordering Problem. *VLDBJ* 6, 3 (1997), 191–208.
[43] Arun Swami. 1989. Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques. In *SIGMOD 1989*. 367–376.
[44] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *PVLDB* 14, 9 (2021), 1640–1654.
[45] Florian Wolf, Michael Brendle, Norman May, Paul R. Willems, Kai-Uwe Sattler, and Michael Grossniklaus. 2018. Robustness Metrics for Relational Query Execution Plans. *PVLDB* 11, 11 (2018), 1360–1372.
[46] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2021. NeuroCard: One Cardinality Estimator for All Tables. *PVLDB* 14, 1 (2021), 61–73.
[47] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *PVLDB* 13, 3 (2019), 279–292.
[48] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *PVLDB* 14, 9 (2021), 1489–1502.

Received July 2023; revised October 2023; accepted November 2023