Hint-QPT: Hints for Robust Query Performance Tuning

Haibo Xiu*, Yang Li*, Qianyu Yang*, Weihang Guo, Yuxi Liu
Pankaj K. Agarwal, Sudeepa Roy, Jun Yang
Duke University, Durham, NC, USA
{haibo.xiu,yang.li,qianyu.yang,weihang.guo,yuxi.liu}@duke.edu,{pankaj,sudeepa,junyang}@cs.duke.edu

ABSTRACT

Query optimizers rely heavily on selectivity estimates to choose efficient execution plans, but inaccuracies in these estimates often result in poor query performance. We introduce **Hint-QPT** (**Hint**s for Robust **Q**uery **P**erformance **T**uning), an interactive tool designed to help users diagnose and improve query performance. Hint-QPT proactively recommends robust plans that are resilient to uncertainty in selectivity estimates, identifies sensitive subqueries for which selectivity estimation errors greatly affect plan quality, and provides intuitive interfaces for targeted selectivity adjustments. Users can either choose the recommended robust plans for execution, or acquire additional statistics on the identified sensitive subqueries to tune query performance. Moreover, Hint-QPT visualizes the alternative execution plans and their costs under uncertainty, helping users to better understand their robustness.

PVLDB Reference Format:
Haibo Xiu, Yang Li, Qianyu Yang, Weihang Guo, Yuxi Liu, Pankaj K. Agarwal, Sudeepa Roy, Jun Yang. Hint-QPT: Hints for Robust Query Performance Tuning. PVLDB, 18(12): 5327 - 5330, 2025.

doi:10.14778/3750601.3750663

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at https://github.com/YoungAndY2m/PARQO_demo_repo.

1 INTRODUCTION

Most database systems have cost-based query optimizers: given a query, they select the execution plan with the lowest cost calculated using estimated selectivities, which determine the size of the intermediate results produced by various subplans. Unfortunately, these selectivity estimates are frequently inaccurate, resulting in suboptimal or even catastrophic runtime performance. When a query performs poorly, users often lack clear insights into the underlying causes. Many database systems, such as PostgreSQL, allow the optimization process to be "hinted" with manually specified plans or subquery selectivities. However, there are an overwhelming number of possibilities, and any one hint can affect the rest of the optimization process in intricate ways. Without guiding principles and supporting tools, query performance tuning is often a frustrating and costly trial-and-error process.

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750663

* These three authors contributed equally.

In this demonstration, we show how Hint-QPT addresses these challenges in query performance tuning, focusing on inefficiencies stemming from unreliable selectivity estimates. Recognizing that inaccurate selectivities are inevitable, Hint-QPT models and incorporates estimation uncertainty as a key factor in the tuning process. Through a friendly interface, Hint-QPT helps users find "robust" plans whose performance remains competitive despite uncertainties, identify "sensitive" selectivities whose estimation errors impact plan optimality the most, further refine plans by selectively acquiring accurate selectivities at runtime, and explore the robustness of alternative plans over uncertain selectivities. In effect, Hint-QPT transforms query performance tuning into a more guided, explainable, and effective process.

In more detail, Hint-QPT supports the following key features:

- **Robust Plan Recommendation.** Leveraging a principled framework for *robust query optimization* in *PARQO* [13], Hint-QPT proactively recommends a *robust* execution plan, which minimizes the expected performance penalty relative to the true optimal plans obtained under exact selectivities. Compared with the default optimizer plan, this robust plan may have a higher cost at the estimated (and potentially inaccurate) selectivities, but on expectation, it delivers competitive performance because the true selectivities often deviate from the estimates.
- **Exploration of Selectivity Estimation Errors.** Given a query, Hint-QPT visualizes the selectivities relevant to its optimization as nodes and edges in a join graph, helping users quickly understand how the subqueries interact. For each selectivity, users can further explore the distribution of errors in its estimate, according to an error model learned in conjunction with the selectivity estimation model.
- **Selectivity and Plan Refinement.** Hint-QPT lets users refine plan choices by selectively adjusting selectivity estimates of subqueries. Optimizing a query involves many relevant selectivities, and adjusting all of them is infeasible. Hint-QPT employs a principled method to highlight a small number of *sensitive selectivity dimensions* [13] — those whose estimate errors have the greatest impact on plan optimality (not necessarily those with the highest uncertainty). Users may choose one (or several) such dimension, override the default estimate, and let the system reoptimize the query. To obtain the overriding value, users can let Hint-QPT acquire the accurate selectivity automatically by executing a counting version of the corresponding subquery.
- **Exploration of Alternative Plans.** Hint-QPT provides an interactive visualization module where users can compare alternative execution plans in terms of both real execution costs and estimated costs under uncertainty. The visualization of real execution costs shows physical operator trees annotated with

operator-level performance, helping users understand the differences among the alternative plans. The visualization of estimated costs under uncertainty compares alternative plans over the space of possible true selectivities for sensitive dimensions, illustrating how robust plans mitigate performance degradation more effectively than default optimizer plans, once we account for the uncertainty in selectivity estimation.

2 BACKGROUND

The key aim of Hint-QPT is to make query performance tuning in the presence of selectivity estimation errors a more guided, explainable, and effective process. To this end, Hint-QPT builds on the PARQO framework [13], which offers a powerful definition of plan "robustness" aligned with the end goal of improving query performance under uncertainty. To quantify uncertainty, we build a statistical model of the selectivity estimation errors, which allows us to cast the problem of finding a robust plan as *stochastic optimization*, and that of identifying sensitive selectivity dimensions as *sensitivity analysis*. More details on PARQO can be found in [13]; here we present only the necessary preliminaries to understand the Hint-QPT demonstration.

Selectivity Dimensions and Error Modeling. Optimization of a query generally depends on a large number of estimates on the sizes of various subqueries. To make error modeling practical, we profile errors in selectivity estimation only for "querylets" [13] — small subquery templates involving up to three joining tables. This profiling can be done over samples of historical queries (which has been done for this demonstration), and can be potentially refreshed by observing estimation inaccuracies in an ongoing query workload. From these error profiles, we construct component error models for all selection and join operations in a query Q, which we refer to as Q's selectivity dimensions. The product of these component models gives the overall error model for Q: suppose $\hat{\mathbf{s}}$ denote the vector of estimated selectivities (one component for each dimensions); then, the error model gives us $f(\mathbf{s}|\hat{\mathbf{s}})$, the distribution of true selectivities $\mathbf{s}$ conditioned on the estimates[1].

Robust Plans. Suppose that the optimizer selects a plan π based on the estimated selectivities $\hat{\mathbf{s}}$, which may deviate from the true selectivities $\mathbf{s}$. Users can specify a $\mathsf{Penalty}$ function to quantify the *penalty* incurred when executing π compared to the true optimal plan under $\mathbf{s}$. For the demonstration, Hint-QPT uses the following function:

$$\mathsf{Penalty}(\pi, \mathbf{s}) = \begin{cases} 0 & \text{if } \mathsf{Cost}(\pi, \mathbf{s}) \leq (1+\tau) \cdot \mathsf{Cost}^{\star}(\mathbf{s}), \\ \mathsf{Cost}(\pi, \mathbf{s}) - \mathsf{Cost}^{\star}(\mathbf{s}) & \text{otherwise.} \end{cases}$$

Here, $\tau = 0.2$ is a tolerance factor: we are fine as long as the plan cost $\mathsf{Cost}(\pi, \mathbf{s})$ is within a factor $(1+\tau)$ of the optimal cost $\mathsf{Cost}^{\star}(\mathbf{s})$, but any amount in excess will be penalized proportionally. Then, the problem of finding a *robust* plan can be defined as the following stochastic optimization problem: given Q, the error model f, and selectivity estimates $\hat{\mathbf{s}}$, find a plan π that minimizes the expected penalty $\mathsf{E}[\mathsf{Penalty}(\pi, \mathbf{S})|\hat{\mathbf{s}}]$ where the true selectivities $\mathbf{S} \sim f(\mathbf{s}|\hat{\mathbf{s}})$.

More details on how to find robust plans can be found in [13]. Although optimizing for robust plans over a distribution is harder than optimizing for a single point in the selectivity space, robust plans offer the benefit that they are expected to perform well despite errors. Furthermore, we find such plans at compile time, without needing to gather additional information and adapt at runtime, making them a good fit for traditional query execution engines.

Sensitive Dimensions. Besides providing a robust plan upfront, Hint-QPT provides another approach to query performance tuning, where the user can devote additional effort to improve the estimates for some selectivity dimensions and reoptimize the query in hope of finding a better plan. The challenge lies in how to narrow down the subset of dimensions to improve. Guided by our penalty-based optimization objective, we employ a global sensitivity analysis method called *Sobol's* [13], which quantifies the variance in $\mathsf{Penalty}(\pi, \mathbf{s})$ across selectivity dimensions by decomposing it into contributions from individual dimension and their interactions. We identify those dimensions with the highest contributions to the total variance as the *sensitive dimensions*, and present them to the users as candidates for further improvement, e.g., using the accurate selectivities acquired by executing the corresponding subqueries.

Hint-QPT's method for prioritizing information gathering to improve query performance is intuitive to understand because it aligns with the users' end goal, and it holds clear advantages over other, less principled, heuristics. For example, one may simply pick a selectivity dimension that is more likely to be wrong; however, correcting its error may not affect the optimal choice at all and can even result in worse plans, because of the complex interactions among various selectivities and cost trade-offs in the optimization process. As another example, one may pick a selectivity dimension with respect to which the plan cost function has a largest partial derivative at $\hat{\mathbf{s}}$; however, having the plan cost sensitive to this dimension does not imply that the plan would become suboptimal when the estimate for dimension is corrected — its cost may still be lower than other alternatives despite its volatility. Overall, because Hint-QPT's framework accounts for both estimate uncertainty (via error modeling) and how errors ultimately affect plan choices (via the penalty-based objective), it can offer better guidance to users than heuristic-based local or limited information.

Implementation. Hint-QPT provides a browser-based graphical user interface for PARQO working on a PostgreSQL Version 16.2 backend (with minor modifications in [12] to enable smoother hinting and interaction). In general, Hint-QPT and PARQO are designed to work on top of any database optimizer that supports injection of selectivity and plan hints.

3 DEMONSTRATION WALKTHROUGH

Our demonstration of Hint-QPT has been pre-loaded with an IMDb database and error profiles collected from sample queries generated from the Join Order Benchmark (JOB) [7]. Below, we provide a walkthrough of the main components of Hint-QPT using Query q17(a) in JOB as an example (Figure 1).

[1]It is worth noting that our error model captures dependencies among multiple selection and join conditions to the extent that they are captured by profiling querylets (which involve combinations of these conditions). For example, the component error model for the join selectivity between tables R and S in Q is based on the errors observed in all querylets involving this join, which may additionally include local selection conditions on R and S and even an additional joining table.

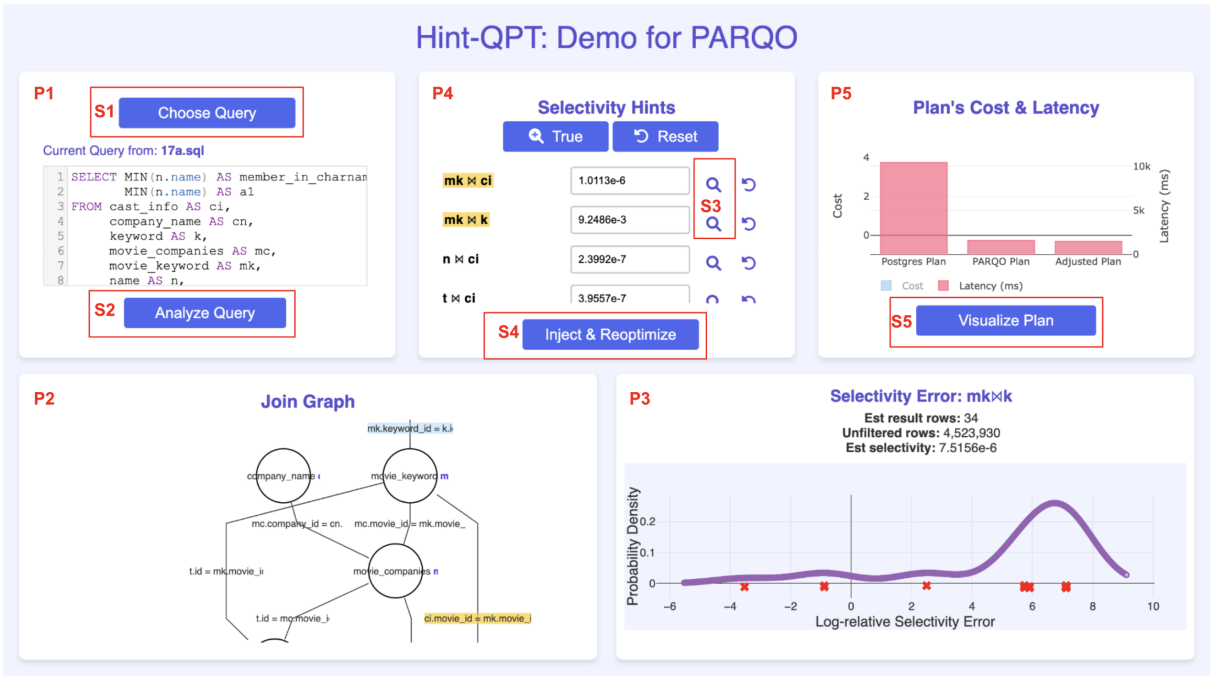

Figure 1: A screenshot of Hint-QPT when tuning the execution performance for Query q17(a) in JOB [7].

3.1 Exploration of Selectivity Errors

Users begin by specifying the query to analyze through the S1 button in the panel marked P1 in Figure 1. For convenience, we have prepopulated the choices using the JOB benchmark queries. In addition, users can select a JOB query template and then specify its parameter settings manually or have them randomly generated. Then, Hint-QPT analyzes the query, generates a join graph in the P2 panel representing the query structure. This interactive visualization, based on SQLVis [8], depicts tables (with their local selections) as nodes and joins as edges. Users can interact with this graph by clicking on a node or edge to explore the uncertainty in its selectivity estimate as modeled by Hint-QPT.

In Figure 1, we have selected the edge of $mk \bowtie k$; therefore, in the P3 panel, Hint-QPT displays some useful information for this selectivity dimension. More precisely, this edge corresponds to the subquery $mk \bowtie \sigma_{keyword='\text{character-name-in-title}'}k$. Hint-QPT shows PostgreSQL's estimate for the result cardinality of this subquery as well as the estimated unfiltered cardinality (i.e., the product of the estimates for $|mk|$ and $|\sigma_{keyword='\text{character-name-in-title}'}k|$, before applying the join condition); PostgreSQL's estimate for this selectivity dimension is the ratio between these two quantities. Hint-QPT also shows the distribution of possible estimation error in this selectivity estimate, as a probability density function learned from error profiling. The actual errors for this querylet observed during the profiling process are plotted as red crosses on the horizontal axis. Users can see from this density function that PostgreSQL tends to overestimate this selectivity by a considerable factor.

3.2 Robust Plan Recommendation

Once the user submits the query, in additional to PostgreSQL's default plan, Hint-QPT also finds a robust PARQO plan aimed at minimizing its expected penalty, as described in Section 2. For this demonstration, we execute both plans immediately for comparison in the P5 panel (ignore "Adjusted Plan" for now; we will discuss it later). In this case, users will see that the PARQO plan runs much faster than PostgreSQL's plan. They can also see that PostgreSQL's estimate of the cost of the PARQO plan is *higher* than that of PostgreSQL's own plan (this comparison view is not shown in Figure 1). This dichotomy is interesting but not surprising, because the estimated selectivities are often wrong in the first place, and PARQO optimizes for overall robustness, as opposed to optimizing for only the cost at the estimated selectivities.

3.3 Selectivity and Plan Refinement

As discussed in Section 2, Hint-QPT selects sensitive selectivity dimensions whose uncertainties have the largest impact on the variance of the plan penalty. In the P2 panel, these nodes or edges are automatically highlighted. For our example query, the two sensitive dimensions identified are $mk \bowtie ci$ and $mk \bowtie k$. Intuitively, reducing uncertainties in the estimates of these two selectivities is likely to give the best payoff in terms of improving the query plan. The P4 panel provides an interface for users to take further actions: all relevant selectivities and their current estimates are shown, and ready to be adjusted; again, the sensitive dimensions are highlighted. Users have two options:

- **Adjust by subquery execution**: Clicking the magnifier icon (S3) next to a selectivity dimension triggers the automatic execution of a counting subquery to obtain the true selectivity value, which then replaces the current setting.
- **Adjust manually**: Users can also directly enter a new setting based on additional knowledge or alternative statistics. For instance, they may run a sampling algorithm to obtain a more accurate estimate, or simply experiment with a different value based on their understanding of the data.

Users can also reset any adjustment by clicking on the undo icon, which restores the selectivity to the original PostgreSQL estimate. The interface also provides "bulk" versions of reset and the function to acquire true selectivities by executing subqueries, though the latter can be time-consuming for complex queries.

After making adjustments, users click the S4 button: Hint-QPT will then inject the specified selectivity values into the query optimizer and have the query re-optimized. The resulting "adjusted plan" is then executed and compared with the default PostgreSQL plan and the robust PARQO plan in the P5 panel. In [13], PARQO experimentally validated that using our principled method for picking sensitive dimensions was more effective than other heuristics; here, using Hint-QPT's interface, users can validate this observation themselves. For our running example, the two most sensitive selectivities, $mk \bowtie ci$ and $mk \bowtie k$, have been set to their true values acquired by subquery execution. In P5, we see that the adjusted plan achieves much better performance than the PostgreSQL plan. Also notable is the observation that the PARQO plan performs comparably to the adjustment plan, even though it does not have the knowledge of the true selectivities acquired at runtime.

3.4 Exploration of Alternative Plans

Besides the high-level comparison of the three alternative plans (PostgreSQL, PARQO, and adjusted) in P5, Hint-QPT also provides an interactive plan exploration interface using Pev2 [2]. The interface parses the `EXPLAIN ANALYZE` output from PostgreSQL and visualizes execution plans as annotated operator trees. Each node represents a physical operator and displays detailed metrics such as estimated cost, actual running time, as well as estimated and actual result sizes. Users can deep-dive into the execution details of the three alternative plans. For nodes with significant selectivity estimation errors, Hint-QPT annotates them to alert users. A dedicated mode highlights only operator durations, making it easy to identify the most time-consuming steps within each plan.

Last but not least, to help users understand plan robustness and the advantages of PARQO's robust plans, Hint-QPT provides a 3D interactive diagram (Figure 2) comparing the estimated costs of the PostgreSQL and PARQO plans over the 2D selectivity subspace spanned by two sensitive dimensions (users can select which two to visualize when there are more than two sensitive dimensions). Each plan is shown as a surface, whose height indicates its costs at a particular selectivity setting. Not all points on the surface are equally important to the comparison because, given the current selectivity estimates, the true selectivities are not uniformly distributed over the entire space. Hence, Hint-QPT uses a color heatmap on the surface, showing more probable regions of selectivities/costs in a darker color. By hovering the cursor over different points on the surface, users can inspect the details under different possible outcomes of the true selectivities.

4 RELATED WORK

Several existing demo systems incorporate visualization tools to aid users in exploring query optimization internals. MOCHA [11] explicitly visualizes the query execution process and enables users to explore alternative plans generated by different physical operators. Jovis [1] also provides an interface for users to manually specify join orders using [9]. Similarly, Spanner [10] includes a

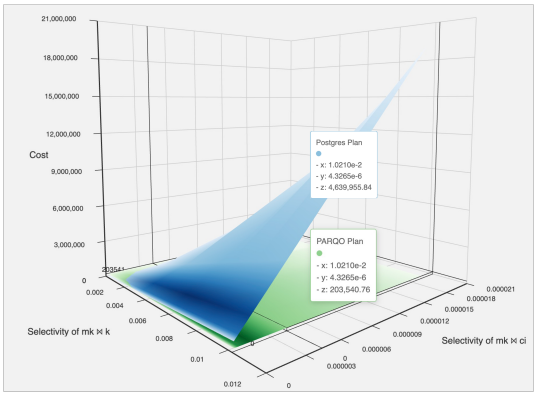

Figure 2: Plan cost surfaces on sensitive subqueries.

plan visualizer and supports user-specified join orders. RobOpt [6] recommends a plan with the lowest sub-optimality risk, predicted by a learned cost model. Different from these systems, Hint-QPT provides a more guided and explainable tool to tune query performance efficiently. Regarding visualization of plan costs, Picasso [5] generates detailed plan diagrams across selectivity spaces, clearly illustrating optimizer plan changes under varying selectivities. Due to the space limit, we refer the reader to comprehensive surveys on robust query optimization for a detailed background [3, 4].

ACKNOWLEDGMENTS

This paper is supported by NSF grant IIS-2402823. Work by Agarwal is also supported by NSF grant CCF-2223870 and by US-Israel Binational Science Foundation Grant 2022131. Work by Yang is also supported by NSF grants IIS-2008107 and II-2211526.

REFERENCES

[1] Yoojin Choi, Juhee Han, Kyoseung Koo, and Bongki Moon. 2024. Jovis: A Visualization Tool for PostgreSQL Query Optimizer. *CoRR* abs/2411.14788 (2024). https://doi.org/10.48550/ARXIV.2411.14788 arXiv:2411.14788

[2] DALIBO. 2016. A VueJS component to show a graphical vizualization of a PostgreSQL execution plan. https://github.com/dalibo/pev2.

[3] Bailu Ding, Vivek Narasayya, and Surajit Chaudhuri. 2024. Extensible Query Optimizers in Practice. *Foundations and Trends® in Databases* 14, 3-4 (2024), 186–402. https://doi.org/10.1561/1900000077

[4] Jayant R. Haritsa. 2024. Robust Query Processing: A Survey. *Found. Trends Databases* 15, 1 (2024), 1–114. https://doi.org/10.1561/1900000089

[5] Naveen Reddy Jayant R Haritsa. 2005. Analyzing plan diagrams of database query optimizers. In *Proceedings of the 31st international conference on Very large data bases. VLDB Endowment.* 1228–1239.

[6] Amin Kamali, Verena Kantere, Calisto Zuzarte, and Vincent Corvinelli. 2024. RobOpt: A Tool for Robust Workload Optimization Based on Uncertainty-Aware Machine Learning. In *Companion of the 2024 International Conference on Management of Data.* 468–471.

[7] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proc. VLDB Endow.* 9, 3 (nov 2015), 204–215. https://doi.org/10.14778/2850583.2850594

[8] Daphne Miedema and George Fletcher. 2021. SQLVis: Visual Query Representations for Supporting SQL Learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC).* IEEE.

[9] Satoshi Nagayasu. 2023. pg_hint_plan. https://github.com/ossc-db/pg_hint_plan.

[10] Google Spanner. 2025. Tune a query using the query plan visualizer. https://cloud.google.com/spanner/docs/tune-query-with-visualizer.

[11] Jess Tan, Desmond Yeo, Rachael Neoh, Huey-Eng Chua, and Sourav S Bhowmick. 2022. MOCHA: a tool for visualizing impact of operator choices in query execution plans for database education. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3602–3605.

[12] Haibo Xiu. 2024. Modified PostgreSQL 16.2. https://github.com/Hap-Hugh/PG16.

[13] Haibo Xiu, Pankaj K. Agarwal, and Jun Yang. 2024. PARQO: Penalty-Aware Robust Plan Selection in Query Optimization. *Proc. VLDB Endow.* 17, 13 (2024), 4627–4640. https://doi.org/10.14778/3704965.3704971