Towards Selecting Informative Alternative Relational Query Plans for Database Education

HUI LI, Xidian University, China
HU WANG, Xidian University, China
SOURAV S. BHOWMICK, Nanyang Technological University, Singapore
ZIHAO MA, Xidian University, China

Off-the-shelf RDBMS typically expose only the query execution plan (QEP) of an SQL query, without presenting information about representative *alternative query plans* (AQPs) considered during plan selection in a *user-friendly* manner. Providing easy access to representative AQPs is valuable in database education, as it helps learners understand the plan choices made by a query optimizer, one of the several important components related to the topic of relational query processing. In this paper, we present a novel problem called *informative plan selection problem* (TIPS) which aims to discover a set of k *informative* AQPs from the underlying plan space so that the *plan informativeness* of the set is maximized. Specifically, we explore two variants of the problem, *batch* TIPS and *incremental* TIPS, to cater to diverse learners. Due to the computational hardness of the problem, we present an approximation algorithm to address it efficiently while providing theoretical guarantees for the results. An extensive experimental study, including feedback from real-world learners and a three-year in-class evaluation of academic outcomes, demonstrates the effectiveness of our solutions for database education.

CCS Concepts: • **Information systems** → **Database query processing**; • **Social and professional topics** → *Computing education.*

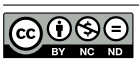
Additional Key Words and Phrases: Query processing and optimization, Database education

ACM Reference Format:
Hui Li, Hu Wang, Sourav S. Bhowmick, and Zihao Ma. 2026. Towards Selecting Informative Alternative Relational Query Plans for Database Education. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 245 (June 2026), 27 pages. https://doi.org/10.1145/3802122

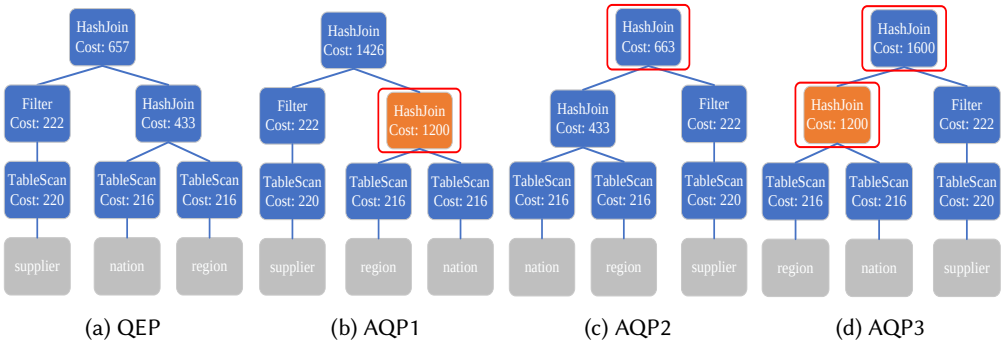
1 Introduction

Given an SQL query, a relational query engine selects its *query execution plan* (QEP) from many *alternative query plans* (AQPs) based on their estimated costs. Although major off-the-shelf relational database systems (RDBMS) expose the QEP of a query to an end user, they do not reveal the AQPs considered by the underlying query optimizer in a *user-friendly* manner. Easy exposition of AQPs is useful in several applications, such as database education [4] and database administration. Although an RDBMS (*e.g., PostgreSQL*) may allow one to manually pose SQL queries with various constraints on *configuration parameters* (*e.g., SET enable_hashjoin = true*) to view the corresponding QEPs containing specific physical operators, this strategy requires not only familiarity with the syntax and semantics of the configuration parameters, but also a clear idea of the AQPs that one is interested in. This is often impractical to expect from end users in practice. For instance, in an academic institution where

Authors' Contact Information: Hui Li, hli@xidian.edu.cn, Xidian University, Xi'an, China; Hu Wang, huwang@stu.xidian.edu.cn, Xidian University, Xi'an, China; Sourav S. Bhowmick, assourav@ntu.edu.sg, Nanyang Technological University, Singapore, Singapore; Zihao Ma, zihao_ma@stu.xidian.edu.cn, Xidian University, Xi'an, China.



ACM 2836-6573/2026/6-ART245
https://doi.org/10.1145/3802122



(a) QEP (b) AQP1 (c) AQP2 (d) AQP3

Fig. 1. Example of QEP and alternative query plans (AQPs).

many learners are taking a database system course for the first time, it is unrealistic to assume that they will manually pose such queries accurately to explore AQPs. Consider the following motivating scenario.

Example 1.1. Lena is currently enrolled in an undergraduate database systems course. She wishes to explore some of the representative AQPs for the following query from the TPC-H benchmark after viewing its QEP as depicted in Fig. 1(a).

```
SELECT s_acctbal, s_name, n_name, s_address, s_phone
FROM supplier, nation, region
WHERE s_acctbal>=5000 and s_nationkey = n_nationkey and
    n_regionkey = r_regionkey;
```

Lena wants to know whether there are AQPs that share similar (*resp.*, different) logical plans and physical operators yet exhibit very different (*resp.*, similar) estimated costs, and, if so, what such AQPs look like. Examples of these AQPs are depicted in Fig. 1(b)-(d). ∎

Clearly, a user-friendly system that can facilitate exploration of *representative* AQPs associated with a given query can greatly aid Lena in answering her queries. Unfortunately, scant attention has been paid by the data management research community to build such a system [4]. In this paper, we present a novel solution for efficient selection of *representative* AQPs for a given SQL query. Our approach not only has received positive feedback from learners (Section 7.1) but also demonstratively improves academic outcomes (Section 7.2).

Selecting a set of *representative* AQPs is technically challenging [4]. First, the plan space can be prohibitively large [7], and many AQPs are neither interesting nor useful to users (*e.g.*, learners). For instance, Lena should not have to inspect all AQPs to learn relational query processing[1]. Thus, exposing all plans is impractical. Instead, we should retrieve a representative subset based on its *informativeness* to end users. However, informativeness is hard to quantify because it can vary across users. In Figures 1(b)-(d), for example, which plans should Lena be shown? Any informativeness measure must account for the plans (including the QEP) a user has already seen, avoiding highly *similar* plans, and reflect the user's interests. Second, scanning all AQPs to find informative ones can be prohibitively expensive. We therefore need efficient techniques to select informative plans. This selection cannot be integrated into the optimizer's enumeration step, since informative AQPs must

[1]Learning relational query processing involves several components, including search algorithms, query rewriting, plan choices, and cost estimation. Our focus here is one component (*i.e.*, plan choices) within this larger set.

be chosen not only based on the QEP a user has seen, but also on learner feedback when available. Third, it may seem natural to allow users to specify $k > 1$ AQPs, as in standard top-k problems. However, this introduces drawbacks in our setting. First, it returns the same set of AQPs for a query regardless of who issues it, though different users may find different plans informative. Second, our user study (Section 7.1) shows that learners often lack confidence in choosing k. Instead, users should be able to inspect AQPs *iteratively*, provide feedback on the usefulness of each plan, and stop once they are satisfied. Thus, k is not only unknown *a priori* but also depends on the plans examined so far. This calls for a flexible framework that supports both *batch* and *incremental* selection of AQPs. Finally, any solution should be sufficiently generic to be implemented on most existing RDBMS.

In this paper, we formalize the aforementioned challenges into a novel problem called *informative plan selection* (TIPS) problem and present a solution to address it[2]. Specifically, given an SQL query and $k \geq 1$ (when k is unspecified it is set to 1), it retrieves k alternative query plans (AQPs) that *maximize plan informativeness*. Such query plans are referred to as *informative* query plans. To this end, we introduce two variants of TIPS, namely *batch* TIPS (B-TIPS) and *incremental* TIPS (I-TIPS), to cater to scenarios where k is specified or unspecified by a user, respectively. We introduce a novel concept called *plan informativeness* to characterize and select informative AQPs. Specifically, it exploits the preferences of real-world users by mapping them to a novel *distance* and *relevance* measures of AQPs *w.r.t.* the QEP or the set of AQPs viewed by a user thus far. These measures are computed by exploiting *differences* between plans *w.r.t.* their topology, physical operators, and estimated cost. This enables us to capture all the relevant dimensions associated with a query plan for any application. Next, given the computational hardness of the TIPS problem, we present a 2-approximation algorithm to efficiently address both variants of the problem. Specifically, it has a linear time complexity *w.r.t.* the number of candidate AQPs. Given that the number of such candidates can be very large for complex queries, we propose two novel *strategies* to prune them efficiently. In this context, we adopt the candidate plan set retrieval framework of the ARENA system [48] to retrieve the candidate plan space for a given SQL query as it allows us to build a generic solution that can easily be realized on most existing RDBMS.

To demonstrate the effectiveness of our AQP selection framework, we deployed it in a database education environment. Specifically, we analyze feedback from real-world learners enrolled in a database systems course, along with their academic performance *w.r.t.* their understanding of the QEP selection process. Our analysis demonstrates the efficiency and effectiveness of our solutions in helping learners understand alternative query plans.

In summary, this paper makes the following contributions.

- We adopt a *learner-centered* approach to characterize the notion of *informative* query plans for database education by engaging real-world learners (Section 2), and present the notion of *plan informativeness* (Section 4) to quantitatively model the informativeness of AQPs.
- We present a novel *informative plan selection* (TIPS) problem to obtain a collection of AQPs for a given SQL query by maximizing *plan informativeness* (Section 3).
- We present approximate algorithms with quality guarantee that exploit plan informativeness and two *pruning strategies* to address the TIPS problem efficiently (Section 5). Our algorithms underpin any framework for exploration of informative AQPs for SQL queries.
- We undertake an exhaustive performance study to demonstrate the superiority of the proposed algorithms over several representative baselines (Section 6).
- We conduct a user study and academic assessment involving real-world learners enrolled in a database systems course to demonstrate the usefulness and effectiveness of our solutions for

[2]A preliminary version of the work was demonstrated in [48].

Table 1. Categorization for *AP*s depending on their distances (*w.r.t.* 3 dimensions) to the QEP.

Plan type	*LogiPln*	*PhyOpr*	*Cost*
AP_I	small	small	small
AP_{II}	small	small	large
AP_{III}	small	large	small
AP_{IV}	large	small	small
AP_V	small	large	large
AP_{VI}	large	small	large
AP_{VII}	large	large	small
AP_{VIII}	large	large	large

pedagogical support (Section 7). These solutions are one piece of a broader effort to address bottlenecks in learning SQL and relational query processing.

The proofs for all theorems and lemmas presented in this work can be found in [47].

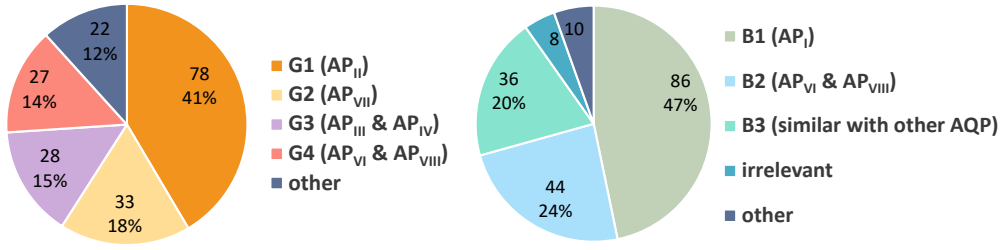
2 Informative Alternative Plans for Education

A key question that any alternative query plan (alternative plan for brevity) selection technique needs to address is as follows: *Which alternative plans are "interesting" or "informative" to a user?* Reconsider Example 1.1. Fig. 1(b)-(d) depict three alternative plans for the query where the physical operator/join order differences are highlighted with red rectangles and significant cost differences are shown using orange nodes. Specifically, all three AQPs have different join orders *w.r.t.* the QEP. However, *AQP1* shares a similar logical plan with QEP but has a substantially higher estimated cost. In contrast, *AQP2* has an estimated cost that is very close to that of QEP. *AQP3* incurs a similarly high cost to *AQP1*. Suppose $k = 2$. Then, among {*AQP1*, *AQP2*}, {*AQP1*, *AQP3*}, and {*AQP2*, *AQP3*}, which one is the best choice? Intuitively, it is reasonable to choose the set that is most "informative" to an individual.

Example 2.1. Reconsider Example 1.1. By examining *AQP1* or *AQP3*, Lena learned that the `HASH JOIN` operator is sensitive to the order of the joined tables, reinforcing concepts she had previously encountered in her classroom lectures. In contrast, *AQP2* shows that altering the join order does not necessarily result in substantially different estimated costs. Lena mistakenly assumed the costs would always differ, based on the examples presented in lectures. Hence, she found *AQP2* informative. Consequently, {*AQP1*, *AQP2*} or {*AQP2*, *AQP3*} are potentially the most informative set for Lena. ∎

2.1 Categorizing Alternative Query Plans

We observe from the above example that the differences between a QEP and AQPs embody valuable information that individuals can garner to supplement their knowledge. Observe that these differences manifest along three dimensions: the *logical plan* (abbr. LogiPln, *i.e.,* the topology of the query plan tree), *physical operators* (abbr. PhyOpr), and the estimated *cost* (Cost) of the plan. Throughout the paper, we consistently use LogiPln to denote the logical plan and PhyOpr to denote physical-operator information. These dimensions determine the informativeness of AQPs. Since a query can have exponentially many AQPs, many of which are similar along these three dimensions, we group them into a concise set of *categories* that characterize informative plans. Specifically, we define eight categories, AP_I to AP_{VIII}, shown in Table 1, based on whether each dimension differs slightly or greatly from the QEP. For instance, a plan that differs only slightly from the QEP



(a) Informative plans (b) Uninformative plans

Fig. 2. Survey results.

in terms of LogiPln and PhyOpr, but exhibits a substantial cost difference, falls into category AP_{II}. These eight categories cover *all* possible combinations of differences from the QEP along the three dimensions.

2.2 Learner-centered Characterization of Informative AQPs

We posit that informativeness of an AQP cannot be determined accurately in practice without considering the interest of end users. Hence, for database education we adopt a learner-centered approach to understand what categories of plans are deemed as *informative* or *uninformative*. Our objective is to examine how learners perceive relationships between plans along the aforementioned dimensions, rather than to exhaustively cover every operator; additional operators, such as `UNION`, naturally fit into this categorization without modification. First, we survey 100 unpaid volunteers who have recently taken a database system course or are currently junior database engineers in industry to determine the informativeness of a set of AQPs. Note that we focus on both groups of volunteers as they cover current learners of a database systems course as well as those who have learned it in the recent past and applied their knowledge in industry. Second, we take the results of the survey as input to classify the aforementioned plan categories into informative or uninformative. Observe that any AQP can be assigned to one of the eight categories. Subsequently, we shall exploit these plan categories to select informative AQPs for any given query.

A key challenge in any human-centered approach is the number of items volunteers must evaluate. Given the exponential number of AQPs for a query, which plans–and how many–should be presented for feedback? Randomly selecting a large number of plans is ineffective: it may not adequately cover the plan categories and, as our interactions with volunteers suggest, large sets discourage feedback by requiring users to browse too many plans. We therefore use AQPs from two different SQL queries on the IMDb dataset [28], each with 3–4 joins and different table scan methods, and ask all volunteers to provide feedback on them. For each query, we show the QEP and 8 AQPs representing the plan categories. These AQPs are heuristically chosen based on their differences from the QEP along the three dimensions. For example, we may select an AQP that exhibits small differences in LogiPln or PhyOpr, but large differences along the remaining dimensions. This strategy covers all plan categories while limiting volunteer effort to reviewing only 18 plans. To mitigate plan-specific factors beyond PhyOpr, LogiPln, and Cost, we present all volunteers with the same representative plan for each category because randomizing plans within a category could therefore bias preferences toward such incidental details rather than the intended category-level differences. Note that our strategy may occasionally select some similar alternative

plans. We deliberately expose these plans to the volunteers to garner their feedback on viewing similar alternative plans.

We ask the volunteers to choose the *most* and *least* informative plans given a QEP and provide justifications for their choices. Specifically, volunteers classify a plan as informative if the *specific* differences between it and the QEP augment, reinforce, or rectify their knowledge related to the plan space and choices (considered by a relational query processor) that they may have acquired through classroom lectures, textbooks, or on-the-job experiences. Note that the aforementioned plan categories are not revealed to volunteers to avoid bias. We received 188 pieces of valid feedback from them (not every volunteer provided effective feedback for all plans). We map the specified informative and uninformative plans to the corresponding plan categories AP_I to AP_{VIII} and count the number of occurrences for each category. Fig. 2(a) reports the distribution of feedback. The eight categories are divided into four groups, *i.e.,* $G_1 - G_4$, depending on the feedback. The group entitled other contains feedback that is too sparse to be grouped as representative of the volunteers. Fig. 2(b) reports the distribution of uninformative plans and categories. B_1 represents plans that are very similar to the QEP (AP_I) and therefore voted uninformative, as they do not convey any useful information. B_2 represents AP_{VI} and AP_{VIII}. Both these categories have large cost differences, as well as at least one other dimension that differs significantly. Given that two of the three dimensions differ significantly (including cost), it is intuitive that these plans do not interest the volunteers. B_3 represents plans that are very similar to other alternative plans viewed by volunteers. In addition, AP_V garnered too little feedback to be deemed informative or uninformative. Moreover, there are eight other pieces of feedback that are orthogonal to the problem addressed in this work (*e.g.,* feedback on the visual interface design). Hence, they are categorized as "irrelevant".

2.3 Informative & Uninformative Plan Categories

In summary, our engagement with volunteers reveals that the majority find alternative plans of categories $AP_{II} - AP_{IV}$ and AP_{VII} informative. However, AP_I, AP_V, AP_{VI}, and AP_{VIII} are not. Observe that the number of feedback responses classifying the latter two as uninformative is substantially higher than those classifying them as informative (G_4 vs. B_2). Hence, we consider them as uninformative. Volunteers also find similar alternative plans uninformative. In the sequel, we shall use these insights to guide the design of algorithms and *pruning strategies* to automatically select informative AQPs.

Remark. It is worth noting that the results of the above human-centered characterization of AQPs into different plan categories are application dependent. In this paper, we focus on database education, and therefore the results in Fig. 2 should be considered within this context. Certainly, for a different application (*e.g.,* database administration), the distribution of plan categories for informative and uninformative AQPs may be different. Nevertheless, the proposed human-centered framework can still be leveraged to determine the distributions.

3 Informative Plan Selection (TIPS) Problem

In this section, we formally define the TIPS problem. We begin with a brief background on relational query plans.

3.1 Relational Query Plans

Given an arbitrary SQL query, an RDBMS generates a *query execution plan* (QEP) to execute it. A QEP consists of a collection of physical operators organized in the form of a tree, namely the *physical operator tree* (operator tree for brevity). Fig. 1(a) depicts an example QEP. Each physical operator, *e.g.,* `SEQUENTIAL SCAN`, `INDEX SCAN`, takes as input one or more data streams and produces an

output one. A QEP explicitly describes how the underlying RDBMS shall execute the given query. Notably, given a SQL, there are many different query plans, other than the QEP, for executing it. For instance, there are several physical operators in a RDBMS that implement a join, *e.g.,* `HASH JOIN`, `SORT MERGE`, `NESTED LOOP`. We refer to each of these different plans (other than the QEP) as an *alternative query plan* (AQP). Figures 1(b)-(d) depict three examples of AQP. It is well-known that the plan space is non-polynomial [7].

3.2 Problem Definition

The *informative plan selection (TIPS) problem* aims to automatically select a small number of alternative plans that maximize *plan informativeness*. These plans are referred to as *informative alternative plans (informative plans/AQPs* for brevity). The TIPS problem has two flavors:

- *Batch TIPS.* A user may wish to view top-k informative plans beside the QEP;
- *Incremental TIPS.* A user may iteratively view an informative plan besides what has already been shown to him or her. Moreover, a user can *rate* each viewed plan *w.r.t.* their informativeness. This rating impacts the succeeding AQP selection.

Formally, we define these two variants of TIPS as follows.

Definition 3.1. Given a QEP π^* and a budget k, the ***batched informative plan selection (B-TIPS) problem*** *aims to find top-k alternative plans Π_{OPT} such that the plan informativeness of the plans in $\Pi_{OPT} \cup \{\pi^*\}$ is maximized,*

$$\Pi_{OPT} = \underset{\Pi}{\operatorname{argmax}}\, U(\Pi \cup \{\pi^*\}) \qquad s.t. \quad |\Pi| = k \quad and \quad \Pi \subseteq \Pi^* \setminus \{\pi^*\}$$

where Π^* denotes the space of all possible plans and $U(\cdot)$ denotes the plan informativeness of a set of query plans.

Definition 3.2. Given a query plan set Π a user has seen so far and the user's preference $r \in \{0, 1\}$ of the current plan, the ***iterative informative plan selection (I-TIPS) problem*** *aims to select an alternative plan π_{OPT} such that the new plan informativeness of the plans in $\Pi \cup \{\pi_{OPT}\}$ is maximized,*

$$\pi_{OPT} = \underset{\pi}{\operatorname{argmax}}\, U_r(\Pi \cup \{\pi\}) \text{ s.t. } \pi \in \Pi^* \text{ and } \pi \notin \Pi$$

where $U_r(\cdot)$ denotes the plan informativeness adjusted according to r.

Here $U_r(\cdot)$ employs the same *MaxMin-style* plan informativeness as $U(\cdot)$, but its reference point is updated in each iteration based on the user's feedback r. The update mechanism is described in detail in Section 4.4 and formalized in Algorithm 1. In the next section, we shall elaborate on the notion of *plan informativeness* $U(\cdot)$ to capture informative plans.

Example 3.3. Reconsider Example 1.1. Suppose Lena has viewed the QEP in Fig. 1(a). She may now wish to view two additional AQPs (*i.e.,* $k = 2$) that are informative. The B-TIPS problem is designed to address it by selecting these AQPs from the entire plan space so that the plan informativeness $U(\cdot)$ is maximized. On the other hand, suppose another learner is unsure about how many alternative plans should be viewed or is dissatisfied with the results returned by B-TIPS. In this case, the I-TIPS problem enables the learner to iteratively select "AQP-at-a-time" and provide a rating *w.r.t.* its informativeness until satisfied. The plan informativeness is maximized by considering the rating from the learner. ∎

Given a plan informativeness measure $U(\cdot)$ and a plan space Π^*, we can find an alternative plan for the I-TIPS problem in $O(|\Pi^*|)$ time. However, the plan space increases exponentially with increasing number of joins and it is NP-hard to find the best join order [25]. This demands efficient solutions especially when there is a large number of joined tables.

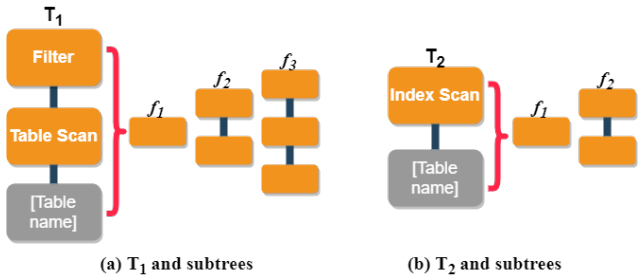


Fig. 3. Plan Tree examples.

Theorem 3.4. *Given a plan informativeness measure $U(\cdot)$, the B-TIPS problem is NP-hard.*

4 Quantifying Plan Informativeness

In this section, we describe the quantification of *plan informativeness* $U(\cdot)$ by exploiting the insights from learner-centered characterization of AQPs in Section 2.

4.1 Lessons from Characterization of AQPs

In Section 2, feedback from volunteers reveals that the selection of informative plans is influenced by the following:

(a) The LogiPln, PhyOpr, and Cost differences of an alternative plan *w.r.t.* the QEP (and other alternative plans) can be exploited to differentiate between informative and uninformative alternative plans. Hence, we need to quantify these *differences* for informative plan selection.
(b) In database education, volunteers typically prefer not to encounter alternative plans that closely resemble the QEP or any plans they have already seen—plans that are similar across all three dimensions: LogiPln, PhyOpr, and Cost. Therefore, the *relevance* of a new alternative plan for a learner should be assessed based on the set of query plans they have viewed so far.

Notably, our survey revealed that learners found plans with minor LogiPln or PhyOpr differences but substantial cost differences (*e.g.,* AP_{II}) particularly informative, which motivates our approach to measuring relevance. We elaborate on the quantification approach to capture these two factors and then utilize them to quantify *plan informativeness*.

4.2 Plan Differences

Broadly, $U(\cdot)$ should facilitate identifying specific AQPs other than the QEP such that by putting them together a learner can reinforce her understanding of the alternative plan choices made by a query optimizer for her query (*i.e.,* informative). This involves effectively quantifying the differences between different plans *w.r.t.* their LogiPln, PhyOpr, and Cost. As $U(\cdot)$ is defined over a set of query plans, we shall first discuss how to quantify these *differences*.

LogiPln Difference. To evaluate the LogiPln difference between a pair of physical operator trees, we need to employ a measure to compute structural differences between trees. The choice of this measure is orthogonal to our framework and any distance measure that can compute the structural differences between trees efficiently can be adopted. By default, we use the *subtree kernel*, which has been widely adopted for tree-structured data [40] to measure the structural difference. Formally, a kernel function [39] is a function measuring the similarity of any pair of objects $\{x, x'\}$ in the input domain $\mathcal{X}$. It is written as $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$, in which ϕ is a mapping from $\mathcal{X}$ to a feature space $\mathcal{F}$. The basic idea of subtree kernel is to express a kernel on a discrete object by a sum

of kernels of their constituent parts. The features of the subtree kernel are *proper subtrees* of the input tree T. A proper subtree f_i comprises node n_i along with all of its descendants. Two proper subtrees are identical if and only if they have the same tree structure. For example, consider T_1 and T_2 in Fig. 3. T_1 has three different proper subtrees, while T_2 only has two subtrees. They share two of them, namely, f_1 and f_2. Note that we do not consider the content of nodes here. Formally, subtree kernel is defined as follows.

*Definition 4.1. Given two trees T_a and T_b, the **subtree kernel** is defined as follows:*

$$\kappa_S(T_a, T_b) = \sum_{n_a \in N(T_a)} \sum_{n_b \in N(T_b)} \Delta(n_a, n_b)$$

where $\Delta(n_a, n_b) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_a) I_i(n_b)$, $\mathcal{F}$ is the feature space that contains all possible proper subtrees, and $I_i(n)$ is an indicator function which determines whether the proper subtree f_i is rooted at node n and $N(T)$ denotes the node set of T.

Since the subtree kernel is a similarity measure, we need to convert it to a distance. Moreover, each dimension may have a different scale. Hence, it is necessary to normalize each dimension following $\kappa(T_a, T_b) = \kappa_S(T_a, T_b) / \sqrt{\kappa_S(T_a, T_a) \cdot \kappa_S(T_b, T_b)}$.

*Definition 4.2. Given two query plans π_a, π_b with corresponding tree structures T_a, T_b, based on the normalized kernel, we can define the **LogiPln distance** between the plans as $s\_dist(\pi_a, \pi_b) = \sqrt{1 - \kappa(T_a, T_b)}$.*

Lemma 4.3. *The LogiPln distance is a metric.*

It is worth noting that the classical approach of computing subtree kernel using suffix tree will lead to quadratic time cost [40]. We advocate that this will introduce a large computational cost since we need to build a suffix tree for each alternative plan at runtime. To address this problem, we first serialize a tree bottom-up, and then use a hash table to record the number of different subtrees. When calculating the subtree kernel, we can directly employ a hash table to find the same subtree. Although the worst-case time complexity is still quadratic, as we shall see in Section 6, it is significantly faster than the classical suffix tree-based approach in practice. We also note that hash-based distance measures (*e.g.*, the Picasso visualizer [21]) primarily rely on plan hashing to group similar plans, whereas our distances are explicitly defined over LogiPln, PhyOpr and Cost. This allows us to reason about learner-perceived differences and to combine these dimensions in a unified *MaxMin* objective.

PhyOpr Difference. In order to further compare the content of the nodes, we adopt the edit distance to quantify PhyOpr differences. Specifically, we first use the preorder traversal to convert a physical operator tree to a string, and based on it we apply the edit distance. Note that the alphabet is the collection of all node contents and the content of each tree node is regarded as the basic unit of comparison. For instance, in Fig. 3, the strings of T_1 and T_2 are ["`Filter`", "`Table Scan`[Table name]"] and ["`Index Scan`[Table name]"], respectively. The edit distance is 2 as they require at least two inserts/replaces to become identical to each other.

By normalizing the edit distance following [30], we define the *PhyOpr distance* as follows.

*Definition 4.4. Given two query plans π_a, π_b and the corresponding strings X_a, X_b, the **PhyOpr distance** between the plans is defined as:*

$$c\_dist(\pi_a, \pi_b) = \frac{2 \cdot edit(X_a, X_b)}{|X_a| + |X_b| + edit(X_a, X_b)}.$$

Notably, join order difference is a special case where the PhyOpr is the same while the order is different. Clearly, Defn. 4.4 is able to differentiate different join orders, *e.g., AP1* and *AP2* in Fig. 1(b) and 1(c).

Remark. There exist several different ways to define the distance between LogiPln and PhyOpr. For example, we could extract a finite-length feature vector for each plan, and then map it to a feature space to calculate the similarity via dot product. We shall investigate this approach in Section 6.

Cost Difference. Finally, because the (time) cost is a real number, we can adopt $L1$ distance and apply Min-Max normalization accordingly.

We are now ready to present the definition of *distance*.

*Definition 4.5. Given two query plans π_i, π_j, the **distance** between them is*

$$dist(\pi_i, \pi_j) = \alpha \cdot s\_dist(\pi_i, \pi_j) + \beta \cdot c\_dist(\pi_i, \pi_j) + (1 - \alpha - \beta) \cdot cost\_dist(\pi_i, \pi_j)$$

where $0 \leq \alpha, \beta \leq 1$ are user-defined weights based on the importance of each component.

4.3 Relevance of an Alternative Plan

The aforementioned distance measures can prevent the selection of an AQP that is similar to plans that have been already viewed. However, these measures individually cannot distill informative plans from uninformative ones. Therefore, we propose a measure called *relevance* (denoted as $rel(\cdot)$), to evaluate the "value" of each plan, so that informative plans receive higher *relevance* scores. Recall the plan categories introduced in Section 2. We convert the differences in each category into a binary form (Fig. 4(a)) where 1 indicates large. We can also represent the informative value of these types (based on learner feedback) in binary form (1 means informative). We can then employ a multivariate function fitting to find an appropriate *relevance function* such that alternative plans that are close to the informative ones receive high scores. With only eight labeled plan types, higher-order models are underdetermined and less interpretable; standard ML regressors (*e.g.,* SVR) do not improve agreement with learner feedback compared to the cubic form. Since eight existing samples are insufficient to fit a polynomial function with an order greater than 3 [26], it should appear in the following form: $\rho_1 \times s + \rho_2 \times c + \rho_3 \times cost + \rho_4 \times s \times c + \rho_5 \times s \times cost + \rho_6 \times c \times cost + \rho_7 \times s \times c \times cost$ where $\rho_1, \ldots, \rho_7$ are the parameters to be fit. By fitting this polynomial, we can define the *relevance* measure as follows.

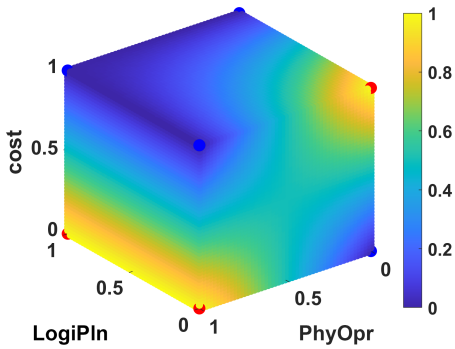
*Definition 4.6. Given the normalized LogiPln distance s_i, PhyOpr distance c_i and cost distance $cost_i$ between an alternative plan π_i and the QEP π^*, the **relevance score** of π_i is defined as $rel(\pi_i) = s_i + c_i + cost_i - s_i \cdot c_i - 2 \cdot s_i \cdot cost_i - 2 \cdot c_i \cdot cost_i + 2 \cdot s_i \cdot c_i \cdot cost_i$.*

The effect of $rel(\cdot)$ is shown in Fig. 4(b). The red vertices represent the plan types that learners consider the most informative, which can effectively be captured by $rel(\cdot)$. In addition, plans close to the informative ones receive high scores accordingly. In fact, each plan π can be referred to as a 3-dimensional vector $\vec{\pi}$ in the space shown in Fig. 4(b). Hence, $rel(\cdot)$ transforms a vector into a scalar value. We shall use $rel(\pi)$ instead of $rel(\vec{\pi})$.

Remark. Note that the distance and relevance scores are different. The former evaluates the differences between plans, while the latter is used to assign a higher score to AQPs that users consider as informative. Any lower-order polynomial cannot satisfy this requirement. More importantly, these measures are generic and orthogonal to the distribution of plan categories and the characterization approach of informative alternative plans in specific applications.

	I	p	cost	*value*
AP_{I}	0	0	0	*0*
AP_{II}	0	0	1	*1*
AP_{III}	0	1	0	*1*
AP_{IV}	1	0	0	*1*
AP_{V}	0	1	1	*0*
AP_{VI}	1	0	1	*0*
AP_{VII}	1	1	0	*1*
AP_{VIII}	1	1	1	*0*

(a) Categories of AQPs



(b) Effect of the fitted function

Fig. 4. Relevance (in (a), the "value" column denotes the relevance score derived from learner feedback).

4.4 Plan Informativeness

Intuitively, through the *plan informativeness* measure, we aim to maximize the minimum relevance and distance of the selected query plan set. To this end, we can exploit measures that have been used in diversifying search results, such as MaxMin, MaxSum, DisC, *etc.* [11, 12]. Based on [13], we utilize the MaxMin model in this work. We adopt the MaxMin criterion because it directly safeguards the *worst-case* informativeness among the selected plans, matching the pedagogical goal of avoiding any "uninformative" outlier. It also provides a simple, interpretable objective with a known approximation guarantee. By contrast, objectives such as MaxSum can be driven by a small number of highly informative plans, while still admitting low-informative ones into the selection.

Suppose the plan set is Π. Then the objective is to maximize $(1-\lambda)\min_{\pi_i \in \Pi} rel(\pi_i) + \lambda \min_{\pi_i, \pi_j \in \Pi} dist(\pi_i, \pi_j)$, where $\lambda > 0$ is a parameter specifying the trade-off between relevance and distance. According to [18], this bi-criteria objective problem can be transformed to finding the distance defined as follows.

Definition 4.7. Given two query plans π_i, π_j, the ***refined distance*** *function is*

$$Dist(\pi_i, \pi_j) = \frac{(1-\lambda)}{2}(rel(\pi_i) + rel(\pi_j)) + \lambda dist(\pi_i, \pi_j).$$

Lemma 4.8. *The distance function defined in Defn. 4.7 satisfies the triangle inequality.*

Consequently, we define the *plan informativeness* $U(\cdot)$ as follows.

Definition 4.9. Given a query plan set Π, the ***plan informativeness*** *is defined as $U(\Pi) = \min Dist(\pi_i, \pi_j)$ where $\pi_i \in \Pi, \pi_j \in \Pi$.*

Incorporating user preferences. Consider U_r in Defn. 3.2. Notably, I-TIPS allows a user to explore *AP* iteratively based on what has been revealed so far. We take into account user preferences in the form of $r \in \{1, 0\}$ (*i.e.,* prefer or not) during the exploration. For example, a user may want to find only those plans with different join orders but not others. This scenario can be easily addressed by I-TIPS, which allows one to mark whether a newly generated plan is preferred. Consider Fig. 4b where each *AP* is represented as a 3-dimensional vector $\vec{\pi}$ *w.r.t.* the QEP ($\vec{\pi^*} = \langle 0, 0, 0 \rangle$). According to Defn. 4.6, a preferred *AP* (as well as its neighbors) should receive higher relevance scores (warmer in the heatmap). Intuitively, finding the category of the preferred *AP* (*i.e.,* $AP_{II} \sim AP_{IV}, AP_{VII}$),

Algorithm 1: Algorithm *update_U*

Input: QEP π^*, user's rating for current AQP r
Output: $U_r(\cdot)$

1 Let $\pi^{(i-1)}$ represents the current AQP.;
2 $AP_{pre} = \textit{argmin}\ dist(AP, \pi^{(i-1)})$ subject to $AP \in \{AP_{II}, AP_{III}, AP_{IV}, AP_{VII}\}$;
3 Let $\Delta_\pi = \epsilon(\vec{AP_{pre}} - \vec{\pi^{(i-1)}})$;
4 update the origin (QEP) : $\vec{\pi^*} = (r == 1)?(\vec{\pi^*} - \Delta_\pi) : (\vec{\pi^*} + \Delta_\pi)$;

say AP_{II} (according to Fig. 4a, we can denote $\vec{AP_{II}} = \langle 0, 0, 1\rangle$), and moving the next plan towards (*resp.,* away from) it if $r = 1$ (*resp., r* = 0), *e.g.,* $\vec{\pi} = \vec{\pi} + \epsilon(\vec{AP_{II}} - \vec{\pi})$ would address this ($\epsilon = 0.05$ by default). However, this strategy can affect only a single AP in the current iteration but not later. Therefore, instead of moving a single AP, we choose to update the origin of the 3-dimensional system, *i.e.,* $\vec{\pi^*} = \vec{\pi^*} - \epsilon(\vec{AP_{pre}} - \vec{\pi})$, where π refers to the latest plan seen so far, and AP_{pre} refers to the category to which π belongs. To find AP_{pre} for a plan π, we only need to perform a nearest neighbor search over the set of $\{\vec{AP_{II}}, \vec{AP_{III}}, \vec{AP_{IV}}, \vec{AP_{VII}}\}$ (each can be encoded according to Fig. 4a) with respect to $\vec{\pi}$. Algorithm 1 formalizes this update of U_r.

5 TIPS Algorithms

As discussed in Section 1, we cannot integrate the selection of informative plans into the optimizer's enumeration step because these plans are chosen based on the QEP and AQPs observed by a learner. Moreover, query optimizers search and filter by cost, whereas we are not merely seeking the lowest-cost plans. Thus, new algorithms are needed for the TIPS problem. We propose I-TIPS and B-TIPS for the two problem variants in Section 3.

We first present two pruning strategies—*group forest-based pruning* (GFP) and *importance-based plan sampling* (IPS)—to enable efficient exploration of the candidate plan space and support queries with many joins. We then describe the I-TIPS and B-TIPS algorithms, which leverage these strategies to select informative plans. It is important to note that the pruning strategies are essential for maintaining acceptable response times even for queries with a small number of joins (*e.g.,* 3–4) that exhibit join-order effects. Such queries may generate thousands of alternative plans due to join-order permutations, access-path choices, and join algorithms [20, 37]. As reported in Section 7.1, GFP keeps the per-AQP wait time around one second.

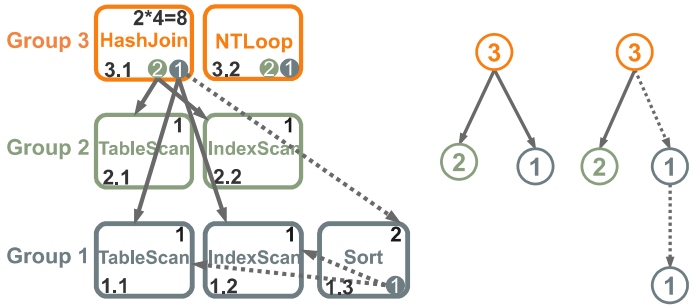
We begin by introducing our framework to retrieve a collection of candidate AQPs Π^* from the underlying RDBMS. These candidate plans are passed on as input to our proposed algorithms.

5.1 Candidate Plan Set Retrieval

To obtain the plan space for a given SQL query (*i.e.,* Π^*), we adopt ORCA [41], which is a modular top-down query optimizer based on the cascades optimization framework [19]. We choose to adopt ORCA because it is a stand-alone optimizer that is portable across different RDBMS systems, thanks to its interaction through a standard XML interface. The only task one needs to undertake is to rewrite the parser that transforms a new format of query plan (*e.g.,* XML format in *SQL Server* and *PostgreSQL*) into the standard XML interface of ORCA. Consequently, this makes TIPS compatible with many RDBMS as a parser can be implemented without modifying the internals of the target RDBMS. Specifically, we have implemented this interface on top of *PostgreSQL* and *Greenplum*.

It is worth noting that the choice of candidate plan set retrieval technique is orthogonal to our work. Other frameworks with similar functionality can also be adopted.

Remark. There are alternative ways to retrieve candidate plans—such as using query hints, altering data statistics. However, such strategies obtain a very limited number of candidates; among them, there are even fewer informative plans. We may also obtain the plans by modifying some physical



(a) MEMO Structure. (b) Group Forest.

Fig. 5. Example of MEMO and Group Forest.

operators or join order of the QEP. This offers no advantage over our approach, restricts candidate plan diversity, and makes it challenging to identify modifications that yield informative plans.

5.2 Group Forest-based Pruning (GFP)

Because the plan space can be very large, traversing all AQPs to find informative plans can hurt efficiency. We therefore need a way to quickly discard uninformative AQPs. As shown in Section 2, AP_{VI} and AP_{VIII} are uninformative, and both have large LogiPln differences from the QEP. A simple approach is to compare each AQP with the QEP and discard those with large differences, but this still requires scanning the entire plan space. To address this, we propose a *group forest-based pruning* (GFP) strategy, which both reduces the number of AQPs and avoids accessing the entire plan space.

Intuitively, the idea is as follows. Reconsider Example 1.1. GFP first groups AQPs according to their *memo-derived group trees*. If a group tree differs significantly from the QEP in terms of LogiPln or Cost (exceeding τ_d or τ_c), all plans in that group—typically AP_{VI}/AP_{VIII}-like—are pruned, while more informative categories such as AP_{II} and AP_{IV} are retained. We elaborate on this strategy now.

Cascade optimizers [19] utilize a structure MEMO (*i.e.,* a forest of operator trees) to track all possible plans. A MEMO consists of a series of *groups*, each of which contains a number of *expressions*. An *expression* is used to represent a logical/physical operator. Fig. 5a shows an example of MEMO, where each row represents a group and the square represents an *expression* in it. Each expression is associated with an *id* in the form of 'GroupID.ExpressionID', shown at the lower left corner (*e.g.,* 3.1, 2.1). The lower right corner of an expression shows the id(s) of the child group(s) to which it points. Different expressions in the same group are different operators that can achieve the same functionality. For example, both 3.1 (`HashJoin`) and 3.2 (`NTLoop`) implement the join operation. Suppose *Group 3* is the *root* group. By traversing down from the expression in the root group, we can get a plan. For example, the id sequence *(3.1, 2.1, 1.1)* represents the plan `HashJoin[TableScan, TableScan]`. If we change one of the expressions, a different plan can be achieved. For example, suppose the child for ❶ of expression 3.1 from 1.1 to 1.3 (as shown by the dotted line in Fig. 5a), we get a different plan `HashJoin[TableScan, Sort[TableScan]]` or `HashJoin[TableScan, Sort[indexScan]]`. The upper right corner of each expression records the number of alternatives with this expression as the root node. For example, the number of expression 1.3 is 2, which includes *1.3⟶1.1* and *1.3⟶1.2*.

Observe that the number of alternatives of expression 3.1 in the root node is 8. That is, there are 8 different plans rooted at expression 3.1. However, there are only 2 tree structures in all these 8 plans, as shown in Fig. 5b. The number in each node represents the id of the group. Since these trees

Algorithm 2: Algorithm IPS

```
Input: QEP π*, collection of all AQPs Π*
Output: sampling of Π*
Π_1 ← ∅;
for π_i in Π* do
    if s_dist(π_i, π*) = 0 then
        Π_1 = Π_1 ∪ {π_i}
    end
end
Π_2 = UniformSampling(Π*);
return Π_1 ∪ Π_2;
```

represent connections between different groups, we collectively refer to them as *group forest*. Each *group tree* in a group forest represents multiple plans with the same structure, and if the LogiPln distance of a group tree and the QEP is larger than a given *threshold* τ_d, *i.e.,* $s\_dist(\pi, \pi^*) \geq \tau_d$, all the corresponding ids can be filtered out. Observe that the number of group trees is significantly smaller than the number of AQPs. Therefore, uninformative plans can be quickly eliminated accordingly. To construct the *group forest*, we only need to traverse MEMO bottom-up, which is extremely fast, *i.e.,* time cost is linear to the height of the constructed tree.

Lastly, observe that if we filter based on s_dist only, informative plans of types AP_{IV} and AP_{VII} can be abandoned. The difference between these two types of plans and AP_{VI} and AP_{VIII} is that the cost is smaller. Hence, we filter using both the LogiPln and Cost differences, *i.e.,* $s\_dist(\pi, \pi^*) \geq \tau_d, cost\_dist(\pi, \pi^*) \geq \tau_c$ where τ_c is a pre-defined *cost threshold*.

5.3 Importance-based Plan Sampling (IPS)

Enumerating the full plan space is prohibitively expensive for queries with many joins (*i.e.,* more than 10 relations). Although sampling can reduce this cost, simply deploying random sampling risks discarding many informative plans. We propose an *importance-based plan sampling* (IPS) strategy that exploits the insights gained from feedback from volunteers in Section 2 to select plans for subsequent processing. The overall goal is to preserve the informative plans as much as possible during the sampling. However, retaining all informative plans will inevitably demand traversal of every AQP to compute plan informativeness. Hence, instead of preserving all informative plans, we choose to keep the most important ones when the number of joined relations in a query is large. We exploit the feedback in Section 2 to sample plans. Observe that AP_{II} plans receive a significant number of votes among all informative plans. Hence, we inject such plans as much as possible in our sample. Also, AP_{II} plans have small LogiPln differences with the QEP. Hence, given a QEP, we can easily determine whether an AQP belongs to AP_{II} or not by finding all alternative plans with the same LogiPln as QEP. These plans are then combined with uniformly random sampled results. For instance, reconsider Example 1.1. IPS preserves AQPs that share the same LogiPln as the QEP (capturing AP_{II}-like plans with large cost differences) and augments them with uniform samples to ensure that informative but rare categories are retained. The pseudocode of the IPS strategy is shown in Algorithm 2.

5.4 I-TIPS Algorithm

Algorithm 3 outlines the procedure to address the I-TIPS problem defined in Defn. 3.2. If the number of joined relations is greater than the predefined *IPS threshold* τ_ℓ, we invoke IPS (Line 2). In addition, if the number of AQPs is greater than the predefined *GFP threshold* τ_g, it invokes the GFP strategy (Line 5), to filter out some uninformative plans early.

Notably, I-TIPS allows a user to explore *AP* iteratively based on what has been revealed so far. We take into account the user preference in the form of rating r during exploration. Algorithm 1

Algorithm 3: Algorithm I-TIPS

Input: QEP π^*, collection of all AQPs Π^*, user's rating r, IPS threshold τ_ℓ, GFP threshold τ_g
Output: an alternative informative query plan $\pi^{(i)}$ during iteration i

```
1  if the number of join tables > τ_ℓ then
2  |   Π* = IPS(π*, Π*);
3  end
4  if |Π*| > τ_g then
5  |   Π* =GFP(π*, Π*) // Group forest-based pruning
6  end
7  update_U(π*, r) // Algorithm 1
8  for π_j ∈ Π* − Π do
9  |   d_j ← Dist(π⃗_j, π⃗*)
10 end
11 Find the plan π^(i) ∈ Π* − Π with the maximum d;
12 Π = Π ⋃{π^(i)};
13 return π^(i)
```

shows the process of updating the plan informativeness, and the procedure strictly follows what we have discussed after Defn. 4.9. Reconsider Example 1.1. After Lena marks AQP_2 as preferred ($r = 1$) plan, I-TIPS shifts the reference point toward its category, guiding subsequent suggestion towards nearby informative categories instead of repeating near-duplicate plans.

5.5 B-TIPS Algorithm

We propose two flavors of the algorithm to address B-TIPS, namely, B-TIPS-BASIC and B-TIPS-HEAP.

B-TIPS-BASIC algorithm. We can exploit Algorithm 3 to address the B-TIPS problem. We first add the QEP π^* to the final set Π. Then iteratively invoke it until there are $k+1$ plans in Π. Each time we just need to add the result of Algorithm 3 to Π. This algorithm has a time complexity of $O(k|\Pi^*|)$.

B-TIPS-HEAP algorithm. Since the minimum distance between each unselected plan in $\Pi^* \setminus \Pi$ and the selected set Π decreases monotonically as $|\Pi|$ increases, we can use a max-heap to filter plans with very small distances to avoid unnecessary computation. Algorithm 4 outlines the procedure. We first apply the two pruning strategies (Line 1). In Line 6, we insert into the heap a tuple containing the ID of an alternative plan π and its distance, defined as the minimum distance between π and the plans in Π. Each time we add a plan to Π, we do not recompute all distances; instead, we only verify whether each stored distance is still valid (Line 17) and update it if necessary (Line 18). At each iteration, we select the alternative plan with the largest minimum distance. For instance, in Example 1.1, when $k = 2$, B-TIPS prefers sets such as $\{AQP_1, AQP_2\}$ or $\{AQP_2, AQP_3\}$, which collectively maximize the minimum relevance-distance trade-off, instead of selecting two highly similar plans. Observe that the worst-case time complexity is also $O(k|\Pi^*|)$. However, it is significantly more efficient than B-TIPS-BASIC in practice (detailed in Section 6).

For a *MaxMin* problem, a polynomial-time approximation algorithm provides a performance guarantee of $\rho \geq 1$, if for each instance of the problem, the minimum distance of the result set produced by the algorithm is at least $1/\rho$ of the optimal set minimum distance. We will also refer to such an algorithm as a ρ-approximation algorithm.

THEOREM 5.1. *B-TIPS is a 2-approximation algorithm if the distance satisfies the triangle inequality (Lemma 4.8).*

Remark. Our algorithms are generic and not limited to database education. First, they are orthogonal to any candidate plan set retrieval strategy. Second, the GFP and IPS strategies—though illustrated with education-focused plan categories—can be easily adapted to other applications (*e.g.,* database administration) with different informative and uninformative plan categories derived from a

Algorithm 4: Algorithm B-TIPS-HEAP

Input: QEP π^*, collection of all AQPs Π^*, budget k, IPS threshold τ_ℓ, GFP threshold τ_g
Output: top-k informative query plans Π

```
1  Lines 1-6 in Algorithm 3;
2  set Heap as a max-heap // record the minimum distance
3  Π ← {π*}
4  foreach π_j ∈ Π* − {π*} do
5      d ← Dist(π*, π_j);
6      Heap.push(tuple(d, π_j));
7  end
8  while |Π| ≠ k + 1 do
9      set tmpRecord is ∅;
10     set optTuple is None;
11     while Heap not empty do
12         tmpTuple ← Heap.pop();
13         if tmpTuple < optTuple then
14             break;
15         end
16         if tmpTuple is not latest then
17             tmpTuple ← Update(tmpTuple);
18         end
19         if tmpTuple > optTuple then
20             optTuple ← tmpTuple;
21         end
22         tempRecord.push(tempTuple);
23     end
24     Π ← Π ∪ {optTuple.π};
       /* plans that are removed from the heap but not in use are put back into the heap   */
25     foreach t in tmpRecord − {optTuple} do
26         Heap.push(t) ;
27     end
28 end
29 return Π \ {π*}
```

human-centered characterization of AQPs. They are also easily adaptable to non-human-centered characterizations of AQPs by modifying the if-statement in Line 3 of Algorithm 2.

6 Performance Study

In this section, we investigate the performance of the TIPS algorithms[3]. In the next section, we report their usefulness and applicability in database education.

6.1 Experimental Setup

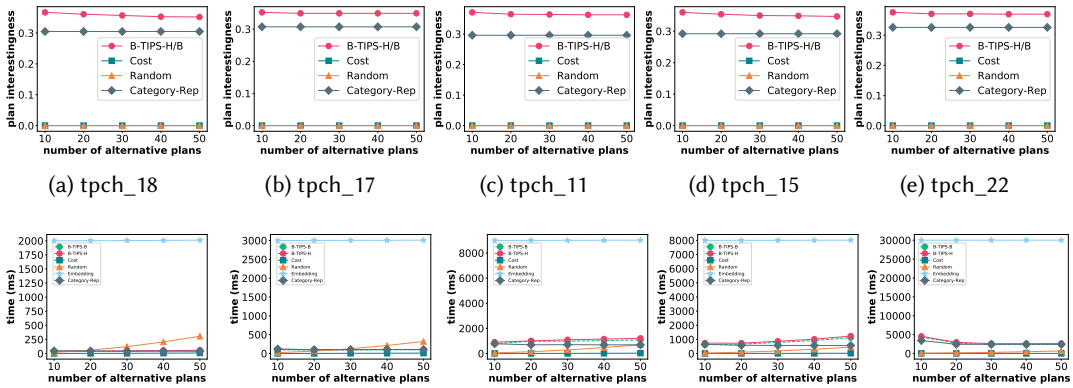
Datasets. We use two datasets. The first is the Internet Movie Data Base (IMDb) dataset [1, 28], containing the two largest tables, cast_info and movie_info have 36M and 15M rows, respectively. The second one is the TPC-H dataset [8]. We use the TPC-H v3 and 1 GB data generated using *dbgen* [8].

Since the number of candidate plans affects the performance of TIPS, which is mainly affected by the SQL statement, we choose different SQL queries to vary the plan size. For IMDb, [28] provides 113 SQL queries. We compute the plan-space size $|\Pi^*|$ for all 113 queries, bin the queries into eight equal-width groups based on $|\Pi^*|$ (*e.g.,* the first group includes queries with plan counts in the range [1500, 2500]), and select the median query in each bin (manually swap it out for a nearby query if its LogiPln is too similar to one that's already been selected). For the TPC-H dataset, there are 22 standard SQL statements. Because of the uneven distribution of the plan space of these queries, it is difficult to find SQL statements that are evenly distributed as in IMDb dataset. We select 5 SQL queries from them for our experiments. To cover more diverse schemas and query patterns, we

[3]The source code is available at https://github.com/ZiHao256/TIPS.

Table 2. Datasets and plan-space sizes.

SQL	$\lvert\Pi^*\rvert$	SQL	$\lvert\Pi^*\rvert$	SQL	$\lvert\Pi^*\rvert$	SQL	$\lvert\Pi^*\rvert$
tpch_18.sql	2070	tpcds_48.sql	956	imdb_6b.sql	2059	imdb_1d.sql	7065
tpch_17.sql	2864	tpcds_34.sql	2357	imdb_2a.sql	3211	imdb_4b.sql	7887
tpch_11.sql	11880	tpcds_26.sql	6177	imdb_5c.sql	3957	imdb_1a.sql	9007
tpch_15.sql	13800	tpcds_13.sql	11309	imdb_2b.sql	4975	imdb_6e.sql	10200
tpch_22.sql	51870	tpcds_47.sql	33769				



(a) tpch_18 (b) tpch_17 (c) tpch_11 (d) tpch_15 (e) tpch_22

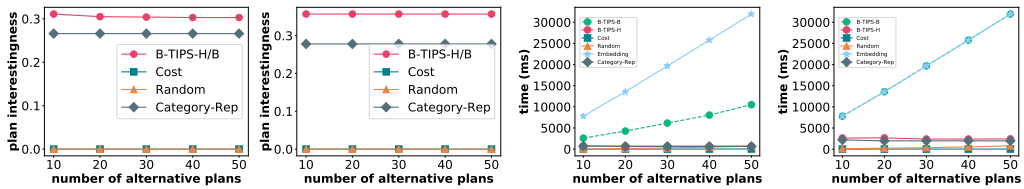
(f) tpch_18 (g) tpch_17 (h) tpch_11 (i) tpch_15 (j) tpch_22

Fig. 6. (a-e) Plan informativeness; (f-j) Running time (for the 5 TPC-H queries in the first column of Table 2).

additionally include five TPC-DS queries (13, 26, 34, 47, 48) as a complementary decision-support workload. These selected queries and their plan space sizes are reported in Table 2.

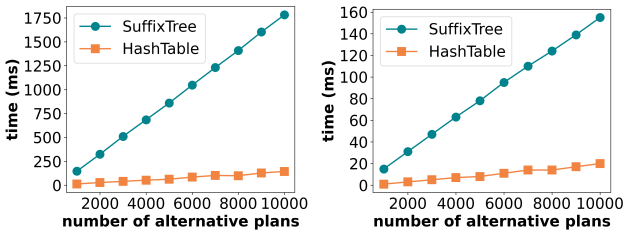
Baselines. We are unaware of any existing informative plan selection technique. Hence, we are confined to comparing the TIPS algorithms with the following baselines. (a) RANDOM: We implemented a random selection algorithm. It is executed repeatedly n times, and each time a result is randomly generated. The best result is returned after n iterations. In our experiment, we set $n = 30$. (b) COST: We implement a *cost-based approach* that returns the top-k plans with the least cost in Π^* besides the QEP. (c) EMBEDDING: We also implemented an embedding-based approach, where we embed each AQP into a vector by feeding its estimated time cost and tree structure into an autoencoder following [3] and then rank the AQPs according to their Euclidean distance *w.r.t.* the QEP. Afterwards, the plan informativeness of the top-k can be computed using Defn. 4.9. (d) CATEGORY-REP: We partition AQPs into eight plan categories by thresholding LogiPln, PhyOpr, and Cost distances into small vs. large, using the same thresholds as GFP, and select one representative per category (the plan with the highest relevance within that category). If $k < 8$, we take the top-k representatives by relevance; if $k \geq 8$, we return all representatives and fill the remaining slots by relevance. This baseline ensures category coverage but does not optimize the global *MaxMin* objective addressed by TIPS.

All algorithms are implemented in C++ on a 3.2GHz 4-core machine with 16GB RAM. We denote B-TIPS-BASIC and B-TIPS-HEAP as B-TIPS-B and B-TIPS-H, respectively. Unless otherwise specified, we set $\alpha = 0.33$, $\beta = 0.33$, $\lambda = 0.5$, $\tau_\ell = 10$, and $\tau_g = 50000$.



(a) tpcds_13 (b) tpcds_47 (c) tpcds_13 (d) tpcds_47

Fig. 7. (a) Plan informativeness; (b) Running time for two TPC-DS queries (13, 47) in Table 2.



(a) Initialization (b) Calculate subtree kernel

Fig. 8. Subtree kernel.

6.2 Experimental Results

Exp 1: Efficiency and effectiveness. We first report the efficiency and effectiveness of different algorithms. For each query in Table 2, we use different algorithms to select k plans and report the plan informativeness U and runtime, shown in Fig. 6(a-e) and 6(f-j), respectively. Figures 6 and 7 report plan informativeness and runtime for five TPC-H queries and two representative TPC-DS queries, respectively, with similar trends observed for the remaining TPC-DS queries. We make the following observations. First, the plan informativeness obtained by RANDOM, COST and EMBEDDING are significantly inferior to that of the B-TIPS algorithms. CATEGORY-REP improves coverage across plan categories but still underperforms B-TIPS because it does not optimize the global *MaxMin* objective. Moreover, the effectiveness of EMBEDDING decreases with respect to k unless the volume of the plan space is large enough, *i.e.*, $|\Pi^*| > 10000$. Second, the running time of B-TIPS-H is closer to COST and RANDOM, significantly faster than B-TIPS-B and EMBEDDING, and stable with increasing k due to the heap-based pruning strategy.

Exp 2: Comparison of strategies for subtree kernel. We compare the efficiency of two subtree kernel computation methods during initialization and execution. The results are reported in Figures 8. Observe that our hash table-based strategy is consistently more efficient.

Exp 3: Impact of relevance. In this set of experiments, we evaluate the relevance measure (Defn. 4.6). We set λ to 0, 0.5, and 1, and use B-TIPS to select 5 AQPs; the results appear in Fig. 9. When $\lambda = 0$ (*resp.*, $\lambda = 1$), U depends only on relevance (*resp.*, distance). Thus, in Fig. 9(a), the selected plans differ greatly from QEP but have low relevance, while in Fig. 9(c), the plans have high relevance but are closer to QEP. By the learner-centric characterization of AQPs in Section 2, both situations are undesirable. When $\lambda = 0.5$, U combines distance and relevance. As shown in Fig. 9(b), the two values are balanced: neither is particularly small, the selected plans are dissimilar yet highly relevant, and the result is more reasonable and informative. We conduct a leave-one-out

PlanId	cost	cost_dist	s_dist	c_dist	relevance
QEP	1390985	0.00	0.00	0.00	0.00
Plan1	100000001401774	1.00	0.54	0.55	0.21
Plan2	100000001393524	1.00	0.47	0.32	0.36
Plan3	100000001406016	1.00	0.50	0.49	0.26
Plan4	1401792	0.00	0.54	0.55	0.79
Plan5	1406468	0.00	0.50	0.55	0.78

(a) the weight of relevance is 0

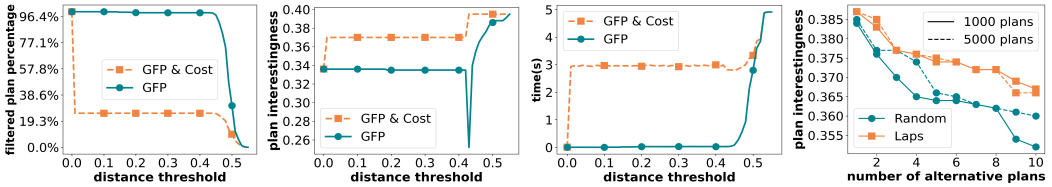
PlanId	cost	cost_dist	s_dist	c_dist	relevance
QEP	1390985	0.00	0.00	0.00	0.00
Plan1	1401793	0.00	0.54	0.55	0.79
Plan2	1405947	0.00	0.50	0.55	0.78
Plan3	1395226	0.00	0.49	0.53	0.76
Plan4	1394466	0.00	0.51	0.51	0.76
Plan5	1401313	0.00	0.52	0.48	0.75

(b) the weight of relevance is 0.5

PlanId	cost	cost_dist	s_dist	c_dist	relevance
QEP	1390985	0.00	0.00	0.00	0.00
Plan1	1401793	0.00	0.54	0.55	0.79
Plan2	1401793	0.00	0.54	0.55	0.79
Plan3	1401793	0.00	0.54	0.55	0.79
Plan4	1401793	0.00	0.54	0.55	0.79
Plan5	1401794	0.00	0.54	0.55	0.79

(c) the weight of relevance is 1

Fig. 9. Effect of relevance.



(a) Effect of pruning (b) Plan informativeness (c) Efficiency (d) Sample 1000/5000 plans

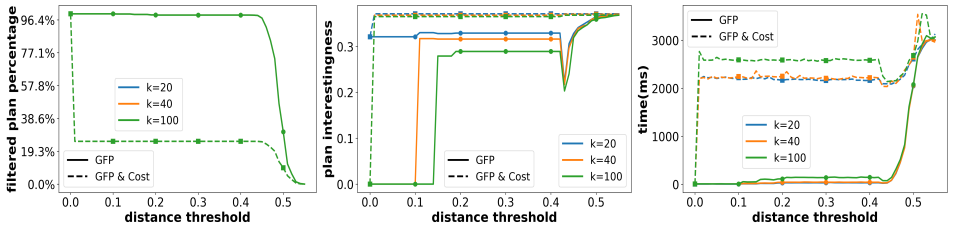
Fig. 10. Pruning strategies: (a–c) effect of GFP (filtered plan percentage, plan informativeness, and efficiency); (d) effect of IPS.

ablation over the eight labeled plan types, comparing the fixed cubic relevance form to standard regressors trained on the same polynomial features. The cubic form achieves the best agreement with learner feedback (MAE 0.225, RMSE 0.237, Spearman $\rho = 0.873$), whereas linear regression and SVR baselines yield higher errors (MAE 0.396–1.048) and lower or negative rank agreement ($\rho \in [-0.86, -0.19]$).

Exp 4: Impact of GFP strategy. We report the performance of the GFP-based pruning strategy, which underpins the system's practical latency and whose impact is evaluated here. Fig. 10(a)-(c) plot the results on *tpch_22.sql*, which has a large plan space. We compare the pruning power, plan informativeness, and efficiency of two variants of GFP, one pruned based only on $s\_dist(\cdot)$ and the other with both $s\_dist(\cdot)$ and $cost\_dist(\cdot)$ (referred to as GFP and *Cost*, respectively). We vary the distance threshold τ_d and set the cost threshold $\tau_c = \tau_d$. Since the distribution of the plans is not uniform, the curves do not change uniformly. Pruning based only on $s\_dist(\cdot)$ removes more plans but degrades plan informativeness. Compared to no GFP (plan informativeness 0.395, time 5s), using GFP with *Cost* (or GFP alone) saves over 40% (or 95%) of runtime while sacrificing less than 6% (or 14%) of plan informativeness, justifying the use of both $s\_dist(\cdot)$ and $cost\_dist(\cdot)$ in GFP and *Cost*.

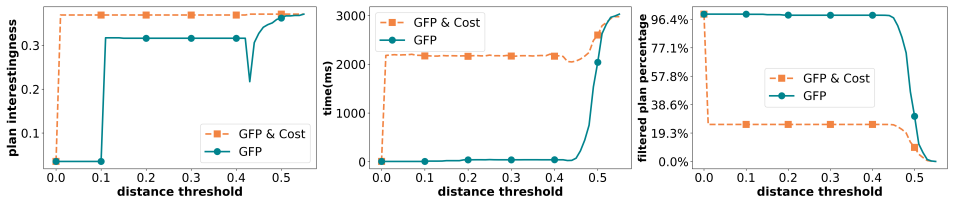
Fig. 11 reports the performance of GFP strategy for different k values. It consistently filters more aggressively but achieves lower informativeness than *GFP+Cost*, and the efficiency-informativeness trade-off remains stable as k increases. Figs. 12 and 13 report additional results for $k = 30$ and $k = 50$, respectively. Observe that they exhibit trends qualitatively similar to those shown in Fig. 11.

Exp 5: Impact of the IPS strategy. We evaluate the IPS strategy from Section 5 using *13c.sql* in [28], which has 11 joins. We create two datasets: (a) a baseline set of n plans ($n = 1000, 2000, \ldots, 5000$) randomly sampled from all available plans; and (b) an IPS-enhanced set constructed by using IPS to collect all alternative plans with the same LogiPln as QEP, add them to n randomly selected



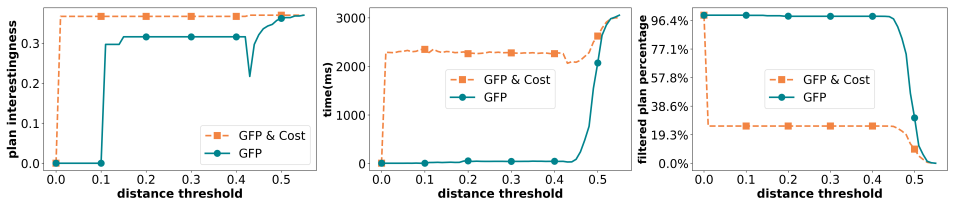
(a) Effect of pruning (b) Plan informativeness (c) Efficiency

Fig. 11. GFP strategy for different k values (*tpch_22.sql*): (a) filtered plan percentage, (b) plan informativeness, and (c) runtime.



(a) Plan informativeness (b) Runtime (c) Filtered plan percentage

Fig. 12. GFP strategy for *tpch_22.sql* with $k = 30$.



(a) Plan informativeness (b) Runtime (c) Filtered plan percentage

Fig. 13. GFP strategy for *tpch_22.sql* with $k = 50$.

alternative plans, and then randomly removing plans to match the baseline's size. We then run B-TIPS on both datasets and compare plan informativeness. Fig. 10(d) shows that IPS typically samples plans with higher informativeness.

7 TIPS for Database Education

We report the usefulness of the TIPS algorithms (collectively referred to as TIPS) for supplementing learning of different plan choices (*i.e.,* AQPs). To this end, we adopt the GUI of ARENA [48] (Fig. 14), which provides a user-friendly interface for retrieving and comparing the QEP and AQPs. Specifically, we conducted a user study and an academic assessment to demonstrate the usefulness of the TIPS algorithms. The various parameters are set to the default (best) values as mentioned in Section 6.

7.1 User Study

We conducted a user study among computer science (CS) undergraduate/postgraduate students who are enrolled in the database course at our university. 51 unpaid volunteers participated in the study. Note that these volunteers are different from those in the survey in Section 2. After consenting to have their feedback recorded and analyzed, we presented a brief tutorial of the ARENA GUI [48],

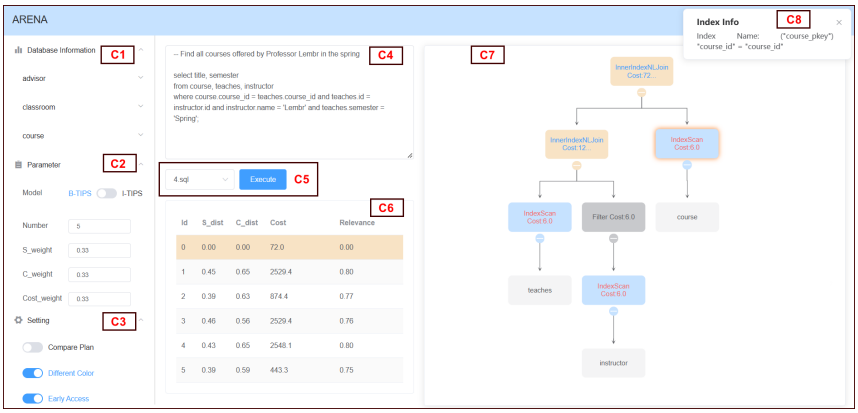


Fig. 14. Screenshot of ARENA interface used in the user study.

describing how learners can use the interface to inspect the QEP and alternative plans. Then they were allowed to explore it for an unstructured time. With GFP enabled, the observed waiting time for each AQP in the study did not exceed approximately 1 second.

US 1: Survey. We presented 30 predefined IMDb queries to all volunteers, who then viewed the returned AQPs for feedback. For our study, we generated AQPs using TIPS, COST, and RANDOM; all plans can be visualized in ARENA GUI. Participants could click any AQP to see it plotted side-by-side with the QEP, showing the tree structure, each node's physical operator, and estimated cost. Volunteers could take as much time as they wished to explore the alternative plans. They then completed a survey with several questions. Where applicable, each subject gave a rating on a 1–5 Likert scale, with higher values indicating stronger agreement with the affirmative statement. We emphasized that survey responses would not affect their grades. Below, we report the key results.

Q1: Is TIPS useful in supporting their learning? Fig. 15a plots the results of the corresponding responses. Observe that 44 participants agree that it is useful for their learning.

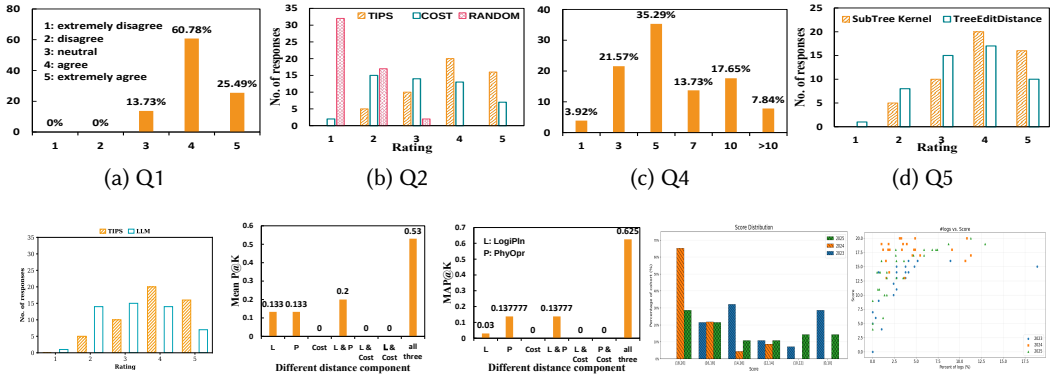
Q2: How well does TIPS/RANDOM/COST help in understanding the alternative plan space? To mitigate bias, the results of TIPS, COST, and RANDOM are presented in random order, and the participants were not informed of the approach used to select the AQPs. Fig. 15b reports the results. In general, TIPS receives the best scores (*i.e.,* 36 out of 51 gave a score over 3).

Q3: Preference for I-TIPS or B-TIPS. Most participants preferred B-TIPS (38 vs. 13), indicating that both I-TIPS and B-TIPS are valuable since learners differ in how they prefer to explore AQPs.

Q4: How many AQPs is sufficient to understand different plan choices? Fig. 15c reports the responses. Most participants felt that 10 or fewer AQPs were sufficient, noting that additional AQPs provided diminishing gains in understanding, which supports the diminishing-return nature of the approximate solution.

Q5: Tree Edit Distance vs. Subtree Kernel. Fig. 15d reports the results. In general, subtree kernel-based structural difference computation strategy receives superior scores (*i.e.,* 36 out of 51 give a score over 3) to tree edit distance-based strategies (*i.e.,* 27 out 51). This justifies the choice of the former as the default LogiPln difference computation technique.

Q6: TIPS vs. LLM-based Learning. Given the popularity of Large Language Models (LLM) in education, we compare TIPS with LLM-based learning (ChatGPT, GPT-4o). Unlike LLM explanations that operate over text descriptions, TIPS exposes concrete optimizer plans and costs, enabling direct comparison of LogiPln/PhyOpr/Cost trade-offs. We follow the *Q2* protocol. After each query, participants rate how well the provided information helps them understand the alternative plan



(a) Q1 (b) Q2 (c) Q4 (d) Q5

(e) Q6 (f) B-TIPS (g) I-TIPS (h) Score distribution (i) Score vs. relative #logs

Fig. 15. User study and academic outcomes.

space. Fig. 15e summarizes the ratings. TIPS outperformed LLM: 36 of 51 TIPS ratings exceeded 3 compared to 20 of 51 for LLM, with a higher average rating (3.92 vs. 3.16) and more ratings concentrated at 4-5 rather than 1-2.

US 2: Impact of different distance components. We explore the impact of different distance components on AQP preferences. We consider various combinations of distance components (x axis in Fig. 15(f) and 15(g)). For each combination, the distance components are equal in weight. These combinations were hidden from the students to mitigate any bias. We choose 3 SQL queries and ask participants to mark and order the top-5 AQPs (from 50 plans returned by our algorithms in each setting) that they most want to see for each query. Note that we fix $k = 50$ here for readability; the system supports larger k and we observe similar trends. We then compare the user-marked list of plans in each case with the results returned by the TIPS algorithms. For B-TIPS, we determine whether the returned result contains the alternative plans that users expect to see and the number of such plans. We do not consider the order in which these plans appear. Therefore, we use *Mean P@k (k=5)* [32] to measure similarity. I-TIPS, on the other hand, returns AQPs iteratively. The order of appearance is important. Therefore, for I-TIPS, we use the *MAP@K (k=5)* [32].

Observe that regardless of the exploration mode, the effect when the three distances (*i.e.,* LogiPln, PhyOpr, and Cost) are considered simultaneously is superior to other combinations. Interestingly, when the weight of cost distance is large, *Mean P@k* and *MAP@k* are both 0. That is, only considering the cost of plans as a factor for selecting AQPs results in potentially suboptimal collection as far as learners are concerned.

US 3: Result differences between I-TIPS and B-TIPS. Lastly, we investigate the differences in the results returned by our proposed algorithms. We collected 80 groups of results from 20 volunteers; each volunteer was requested to use both I-TIPS and B-TIPS and view 10 AQPs. For each AQP returned in I-TIPS, volunteers can mark whether it is useful to them (*i.e.,* rating r) and the AQPs returned subsequently take this feedback into account. We observe that, on average, 56.07% of the returned AQPs are different in both solutions.

7.2 Academic Outcomes

We conducted a three-year longitudinal study across three cohorts of an undergraduate computer science degree program. The *TIPS-based pedagogical model* was introduced to each of the cohorts in

February 2023, *February 2024* and *February 2025* by the end of March, respectively, with courses ending in mid-June and a final paper-based exam held later that month. Each exam included a 20-mark question assessing students' understanding of alternative plans, including (a) other AQPs that can also answer the query; (b) whether a plan is better than another; and (c) what plans the optimizer will never choose. The question set is available in [47]. We report *relative* (within-cohort normalized) counts: within each cohort, score-bin counts are divided by cohort size so that bins sum to 100%, and log-score plots normalize each student's logs by the cohort's total logs. Fig. 15 presents these results.

Each cohort consisted of about 30 undergraduate students. Over the three-year study, we varied how the TIPS-based pedagogical model was integrated into teaching: in 2023, TIPS was introduced without required practice; in 2024, students were encouraged to practice extensively on the ARENA GUI, with usage contributing to their lab homework grade (the final grade consists of final exam score, code-based project score, and lab homework score); and in 2025, practice was optional and ungraded to assess voluntary adoption. To ensure unbiased evaluation, different teaching assistants—unaware of TIPS—graded the final exams for each cohort, enabling us to address the following research questions:

RQ1: Do TIPS help to improve academic outcomes? Students in 2024 performed best, with over half scoring above 18, compared to fewer than one third in 2025 and none in 2023, indicating superior overall performance for the 2024 cohort, followed by those in 2025. We conducted a one-way ANOVA to determine whether the differences between the score distributions of the three cohorts are significant. The test resulted in a p-value of 0.000008, which indicates that the different pedagogical model of TIPS significantly affects the scores obtained by students across the three cohorts.

RQ2: Is there any correlation between the user logs and their outcomes? As illustrated in Figure 15i, quiz scores generally increase with the percentage of cohort logs, exhibiting a pattern of diminishing returns. The improvement is particularly pronounced for the 2023 cohort, where scores rise rapidly with increasing log percentages, likely due to their relatively limited number of logs compared to the other cohorts. However, once the proportion of logs reaches approximately 5–10%, further increases yield little additional gain in scores. Because the 2024 cohort were encouraged to use TIPS, their average number of logs is correspondingly higher than those of the 2023 and 2025 cohorts. Given the positive relationship between the number of logs and performance, this also accounts for why their score distribution in Figure 15h is the strongest among the three cohorts.

Remark. We analyzed the login and query logs of TIPS and found that database course students have used it frequently over the past two years. Their queries often involve complex nested subqueries and multi-table joins (more than three tables), indicating that students actively rely on our framework to better understand plan choices for complex queries.

7.3 Beyond Database Education

Finally, we briefly describe how our framework can be used beyond database education. Specifically, we highlight two practical use cases where it benefits DBAs and junior database engineers.

Slow-query diagnosis. In traditional tuning, DBAs often spend considerable effort examining minor variants of the same inefficient plan. By suppressing AQPs that are structurally or operator-wise similar to the QEP, TIPS enables a DBA to explore queries that are slow by identifying AQPs from a different "quadrant" of the search space (*e.g.,* plans with a substantially different join order but comparable estimated cost). If such a structurally distant AQP performs better in practice, it indicates that the optimizer's cost model may be biased toward a particular plan shape (*e.g.,* left-deep trees) that is suboptimal for the current data distribution. Moreover, by exposing plans

that differ significantly in their physical operators, TIPS can also help DBAs determine whether a performance bottleneck stems from a specific implementation (*e.g.,* a Nested Loop Join causing I/O thrashing) rather than the query logic itself. Because our framework presents the DBA with a "candidate list" of distinct physical strategies to test, it can potentially accelerate the process of identifying an effective solution.

Onboarding junior engineers. TIPS can serve as a valuable tool for onboarding junior database engineers. By filtering out AQPs that closely resemble the QEP, it immediately exposes juniors to high-contrast examples, allowing them to examine why the optimizer favors a Hash Join-based pipeline over a structurally different Sort-Merge-based alternative. Additionally, by reviewing k plans that vary across both LogiPln and PhyOpr, junior DBAs can develop an understanding of how different workload patterns (*e.g.,* OLTP vs. OLAP) translate into distinct plan shapes.

8 Related Work

Our work is closely related to the problem of diversified top-k query processing, which seeks to compute the k most relevant results for a user query while explicitly incorporating diversity into the ranking objective. This problem has been extensively investigated across a broad range of settings, including diversified keyword search in databases [52], textual document collections [2], and graphs [17]; diversified top-k pattern matching [14]; diversified clique enumeration [51]; and diversified structural search in graphs [24], among others [46, 53]. Furthermore, [11] provided a comprehensive survey of methodologies for diversifying query results, Ting and Jin [10] study the computational complexity of diversification problems, and Gollapudi and Sharma [18] employ an axiomatic framework to analyze diversification systems. Nevertheless, these techniques are not directly applicable to our setting, as they do not address the informativeness of query execution plans within a relational database management system (RDBMS).

Databases and Structured Query Language (SQL) are core topics in Software Engineering and Computer Science curricula [42]. Because students struggle with SQL [34], many tools have been developed to help them understand complex statements [9, 23, 27, 29, 33, 35] and learn to write correct, efficient queries [6, 22, 36, 43, 49]. Other work focuses on visualizing SQL queries [15, 16, 29, 35] and query plans [38]. Far less research targets technologies for learning relational query processing [4, 5, 21, 31, 45, 50]. NEURON [31] and LANTERN [50] generate natural language descriptions of QEPs. MOCHA [44] supports learner-friendly interaction and visualization of how alternative physical operators affect a chosen QEP, but assumes learners already know which operators they want to explore. Picasso [21] visualizes different QEPs and their costs across the selectivity space. Our work complements these systems by enabling exploration of informative AQPs.

9 Conclusions and Future Work

This paper presents the informative plan selection (TIPS) problem, one piece of a larger problem of technology-enabled learning of relational query processing. We propose efficient approximate algorithms (B-TIPS and I-TIPS) that maximize the plan informativeness of the selected plans. These algorithms are integrated into our tool called TIPS for database education. Experimental studies, feedback from real-world learners, and analysis of academic performance demonstrate the effectiveness of our solutions. As part of future work, we wish to facilitate learning of various other components related to relational query processing such as the query plan selection algorithm, query rewriting, and cost estimation. As mentioned above, the TIPS algorithms can be also adapted for database administration, which is an interesting direction for future work.

References

[1] 2024. Internet Movie Data Base (IMDb). https://www.imdb.com/.

[2] Albert Angel and Nick Koudas. 2011. Efficient diversity-aware search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*. ACM, 781–792.

[3] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A Simple but Tough-to-Beat Baseline for Sentence Embeddings. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SyK00v5xx

[4] Sourav S Bhowmick and Hui Li. 2022. Towards Technology-Enabled Learning of Relational Query Processing. *IEEE Data Engineering Bulletin* 45, 3 (2022).

[5] Sourav S. Bhowmick and Hui Li. 2025. Experience Report on Using LANTERN in Teaching Relational Query Processing. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE TS 2025, Pittsburgh, PA, USA, 26 February 2025 - 1 March 2025*. ACM, 123–129.

[6] Peter Brusilovsky, Sergey Sosnovsky, Michael V. Yudelson, Danielle H. Lee, Vladimir Zadorozhny, and Xin Zhou. 2010. Learning SQL Programming with Interactive Tools: From Integration to Personalization. *ACM Trans. Comput. Educ.* 9, 4 (Jan. 2010), 19:1–19:15.

[7] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 34–43.

[8] Transaction Processing Performance Council(TPC). 2021. TPC-H Version 2 and Version 3. http://www.tpc.org/tpch/.

[9] Jonathan Danaparamita and Wolfgang Gatterbauer. 2011. QueryViz: helping users understand SQL queries and their patterns. In *Proceedings of the 14th International Conference on Extending Database Technology*. 558–561.

[10] Ting Deng and Wenfei Fan. 2013. On the Complexity of Query Result Diversification. *Proc. VLDB Endow.* 6, 8 (2013), 577–588.

[11] Marina Drosou and Evaggelia Pitoura. 2010. Search result diversification. *ACM SIGMOD Record* 39, 1 (2010), 41–47.

[12] Marina Drosou and Evaggelia Pitoura. 2012. Disc diversity: result diversification based on dissimilarity and coverage. *arXiv preprint arXiv:1208.3533* (2012).

[13] Marina Drosou and Evaggelia Pitoura. 2013. POIKILO: A Tool for Evaluating the Results of Diversification Models and Algorithms. *Proc. VLDB Endow.* 6, 12 (Aug. 2013), 1246–1249.

[14] Wenfei Fan, Xin Wang, and Yinghui Wu. 2013. Diversified Top-k Graph Pattern Matching. *Proc. VLDB Endow.* 6, 13 (2013), 1510–1521.

[15] Wolfgang Gatterbauer. 2024. A comprehensive tutorial on over 100 years of diagrammatic representations of logical statements and relational queries. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 5387–5392.

[16] Wolfgang Gatterbauer, Cody Dunne, HV Jagadish, and Mirek Riedewald. 2022. Principles of Query Visualization. *arXiv preprint arXiv:2208.01613* (2022).

[17] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. 2008. Keyword proximity search in complex data graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*. ACM, 927–940.

[18] Sreenivas Gollapudi and Aneesh Sharma. 2009. An Axiomatic Approach for Result Diversification. In *Proceedings of the 18th International Conference on World Wide Web* (Madrid, Spain) *(WWW '09)*. Association for Computing Machinery, 381–390.

[19] Goetz Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.* 18, 3 (1995), 19–29.

[20] Goetz Graefe and William J McKenna. 1993. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of IEEE 9th international conference on data engineering*. IEEE, 209–218.

[21] Jayant R. Haritsa. 2010. The Picasso Database Query Optimizer Visualizer. *Proc. VLDB Endow.* 3, 2 (2010), 1517–1520.

[22] Yihao Hu, Amir Gilad, Kristin Stephens-Martinez, Sudeepa Roy, and Jun Yang. 2024. Qr-hint: actionable hints towards correcting wrong sql queries. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[23] Yihao Hu, Zhengjie Miao, Zhiming Leong, Haechan Lim, Zachary Zheng, Sudeepa Roy, Kristin Stephens-Martinez, and Jun Yang. 2022. I-Rex: An interactive relational query debugger for SQL. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*. 1180–1180.

[24] Xin Huang, Hong Cheng, Rong-Hua Li, Lu Qin, and Jeffrey Xu Yu. 2013. Top-K Structural Diversity Search in Large Networks. *Proc. VLDB Endow.* 6, 13 (2013), 1618–1629.

[25] Toshihide Ibaraki and Tiko Kameda. 1984. On the optimal nesting order for computing n-relational joins. *ACM Transactions on Database Systems (TODS)* 9, 3 (1984), 482–502.

[26] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. 2013. *An introduction to statistical learning*. Vol. 112. Springer.

[27] Andreas Kokkalis, Panagiotis Vagenas, Alexandros Zervakis, Alkis Simitsis, Georgia Koutrika, and Yannis E. Ioannidis. 2012. Logos: a system for translating queries into narratives. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*. ACM, 673–676.

[28] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215.

[29] Aristotelis Leventidis, Jiahui Zhang, Cody Dunne, Wolfgang Gatterbauer, HV Jagadish, and Mirek Riedewald. 2020. QueryVis: Logic-based diagrams help users understand complicated SQL queries faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2303–2318.

[30] Yujian Li and Bi Liu. 2007. A Normalized Levenshtein Distance Metric. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 6 (2007), 1091–1095.

[31] Siyuan Liu, Sourav S Bhowmick, Wanlu Zhang, Shu Wang, Wanyi Huang, and Shafiq Joty. 2018. NEURON: Query Optimization Meets Natural Language Processing For Augmenting Database Education. *arXiv preprint arXiv:1805.05670* (2018).

[32] Brian McFee and Gert Lanckriet. 2010. Metric learning to rank. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Haifa, Israel) *(ICML'10)*. Omnipress, Madison, WI, USA, 775–782.

[33] Zhengjie Miao, Sudeepa Roy, and Jun Yang. 2019. Explaining wrong queries using small examples. In *Proceedings of the 2019 International Conference on Management of Data*. 503–520.

[34] Daphne Miedema, Efthimia Aivaloglou, and George Fletcher. 2022. Identifying SQL misconceptions of novices: Findings from a think-aloud study. *ACM Inroads* 13, 1 (2022), 52–65.

[35] 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, 1–9.

[36] Daphne Miedema, George Fletcher, and Efthimia Aivaloglou. 2022. Expert Perspectives on Student Errors in SQL. *ACM Trans. Comput. Educ.* 23, 1 (Dec. 2022), 11:1–11:28.

[37] Raghu Ramakrishnan and Johannes Gehrke. 2003. *Database management systems (3. ed.)*. McGraw-Hill.

[38] Daniel Scheibli, Christian Dinse, and Alexander Boehm. 2015. QE3D: Interactive Visualization and Exploration of Complex, Distributed Query Plans. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 877–881.

[39] John Shawe-Taylor and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

[40] Alex Smola and S.v.n. Vishwanathan. 2003. Fast Kernels for String and Tree Matching. In *Advances in Neural Information Processing Systems*, Vol. 15.

[41] Mohamed A. Soliman, Lyublena Antova, Venkatesh Raghavan, Amr El-Helw, Zhongxian Gu, Entong Shen, George C. Caragea, Carlos Garcia-Alvarado, Foyzur Rahman, Michalis Petropoulos, Florian Waas, Sivaramakrishnan Narayanan, Konstantinos Krikellas, and Rhonda Baldwin. 2014. Orca: a modular query optimizer architecture for big data. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*. ACM, 337–348.

[42] Toni Taipalus and Ville Seppänen. 2020. SQL Education: A Systematic Mapping Study and Future Research Agenda. *ACM Transactions on Computing Education* 20, 3 (Sept. 2020), 1–33.

[43] Toni Taipalus, Mikko Siponen, and Tero Vartiainen. 2018. Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education* 18, 3 (Sept. 2018), 1–29.

[44] 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.

[45] Yuan Tian, Jonathan K. Kummerfeld, Toby Jia-Jun Li, and Tianyi Zhang. 2024. SQLucid: Grounding Natural Language Database Queries with Interactive Explanations. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 12, 20 pages.

[46] Bin Wang, Rui Zhu, Xiaochun Yang, and Guoren Wang. 2018. Top-K representative documents query over geo-textual data stream. *World Wide Web* 21, 2 (2018), 537–555.

[47] Hu Wang, Hui Li, Sourav S Bhowmick, and Zihao Ma. 2026. Towards Selecting the Informative Alternative Relational Query Plans for Database Education. arXiv:2210.13722 [cs.DB] https://arxiv.org/abs/2210.13722

[48] Hu Wang, Hui Li, Sourav S Bhowmick, and Baochao Xu. 2023. ARENA: Alternative Relational Query Plan Exploration for Database Education. In *Companion of the 2023 International Conference on Management of Data*. 107–110.

[49] Jinshui Wang, Shuguang Chen, Zhengyi Tang, Pengchen Lin, and Yupeng Wang. 2024. False Positives and Deceptive Errors in SQL Assessment: A Large-Scale Analysis of Online Judge Systems. *ACM Transactions on Computing Education* 24, 3 (Sept. 2024), 1–23.

[50] Weiguo Wang, Sourav S. Bhowmick, Hui Li, Shafiq R. Joty, Siyuan Liu, and Peng Chen. 2021. Towards Enhancing Database Education: Natural Language Generation Meets Query Execution Plans. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 1933–1945.

[51] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2015. Diversified top-k clique search. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*. IEEE Computer Society, 387–398.

[52] Feng Zhao, Xiaolong Zhang, Anthony K. H. Tung, and Gang Chen. 2011. BROAD: diversified keyword search in databases. *Proc. VLDB Endow.* 4, 12 (Aug. 2011), 1355–1358.
[53] Rui Zhu, Bin Wang, Xiaochun Yang, Baihua Zheng, and Guoren Wang. 2017. SAP: Improving Continuous Top-K Queries Over Streaming Data. *IEEE Trans. Knowl. Data Eng.* 29, 6 (2017), 1310–1328.

Received October 2025; revised January 2026; accepted February 2026