On Exploiting Query Plan Logs for Query Optimization

by

Satyanarayana R Valluri, Kamalakar Karlapalem, Arvind Hulgeri

Report No: IIIT/TR/2009/154

Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
September 2009
On Exploiting Query Plan Logs for Query Optimization

Satyanarayana R Valluri Arvind Hulgeri Kamalakar Karlapalem
satya@research.iiit.ac.in aru@persistent.co.in kamal@iiit.ac.in

Abstract
Modern day query optimizers employ complex heuristics and optimizations to reduce the time taken for optimization. Query optimization has become complicated and hence difficult to understand. In this paper, we aim to answer the question: can we determine the optimal query execution plan of a query by bypassing the plan enumeration phase of query optimization? Empirically, we deduce the relationship between the query and its optimal execution plan by executing a large number and large variety of queries. We show by means of experiments that the relationship discovered can be used to determine the optimal query execution plan of a query with high accuracy.

1 Introduction
Query optimization addresses the problem of determining the optimal query execution plan (hence forth referred to as optimal plan) for a given query. The worst case complexity of the dynamic programming algorithm [21] is exponential in the number of relations [24]. Query optimizers find the optimal plans by traversing through the plan space, assigning cost values to each of these plans using a cost model and selecting the least cost plan. Since the possible number of plans for a given query are large number (typically exponential in the number of relations of the query), plan enumeration is expensive. Hence, eliminating the plan enumeration, i.e., avoiding traversing through the possible query plans to find the optimal plan\(^1\) will reduce the complexity of the query optimization. However, the cost of the plans discovered without plan enumeration should be comparable to that of the cost of the optimizer plans\(^2\).

The question that is addressed in this paper is: “Can we build a system which can bypass the plan enumeration phase of query optimization to determine the optimal plan of a query?” However, the system might use or invoke the query optimizer before the plan discovery phase to study the nature of the optimal plans. However, once the characteristics of the optimal plans are known, we should not require calling the optimizer to enumerate the plan space. The answer to the above question can be obtained by answering the following two questions:

- What are the characteristics and the representative features of the optimal plan space?
- How can we use the representative features of the optimal plan space to determine the optimal plan of a query?

\(^1\)Optimal Plan: Least cost plan which may or may not be same as the optimizer plan\(^2\).
\(^2\)Optimizer Plan: Plan that is selected by the query optimizer to be executed on the database by the database system.
Based on our empirical analysis, we found that the optimal plan space can be characterized by a set of join order templates (section 2). A small set of join order templates cover the optimal plans of a large number of queries. For example, in experiment 2 of Section 2, it is shown that in a workload of 100k queries, about 50 join order templates cover the optimal plans of 75k queries. These templates can be used to discover the optimal plan of a new query without plan enumeration (section 3). Determination of the optimal plan of a query involves evaluating the cost of executing the query using a set of query execution plans, but it does not involve enumerating the plan space. We also assume that the query optimizer is a black box and hence our system can be built on the top of a DBMS without modifying the internals.

This paper reports the observations and the results we obtained in answering the questions mentioned previously. We built a proof-of-concept system that illustrates the utility of our approach (i.e., finding optimal plan without plan enumeration).

Figure 1: X and Y axes denote the plan diagram grid points. (a) Plan Distribution for 2D Plan Diagram Query Space on 32×32 grid. Two plans that cover the entire space are denoted by △ and ■ which cover 784 and 240 queries respectively. (b)Plan Logs Generation Query Sample for 2D Plan Diagram Query Space on 32×32 grid. □ denotes the queries used for plan logs generation. ✶ denote the queries for which better plan is discovered. (c)Refined 2D Plan Diagram on 32×32 grid. Two plans: △ and ■ cover 912 and 112 queries respectively.

[20] analyzes the plan space using plan diagrams. They show diagrammatically the coverage of the plan space for 1D and 2D queries (queries that are generated by varying the selection predicate of 1 or 2 attributes respectively, see section 4.2 for a more detailed description). We applied our algorithm on plan diagram scenario where we generated the logs generation queries (the queries for collecting the logs of the optimal query plans) and the evaluation queries (plans used to test the effectiveness of the plan discovery) based on 2D and 3D query templates. We call this plan diagram query space. We also tested our approach on a general scenario where these queries need not be generated based on a plan diagram query space, but can be any random queries. We call the general scenario as generic query space.

Showing that our approach works in a constrained scenario like plan diagram query spaces, provides minimum proof-of-concept of our approach. In other words, if our approach is not effective in a constrained query space like plan diagram query space, it is highly improbable that it will work effectively in a generic query space which is more complex. Also, recent works related to plan diagrams reduce the number of plans in the plan diagrams [6, 2] and find robust plans in the plan diagram [7]. The join order templates collected using a subset
of queries in the plan diagram represents a summary of the optimal plans of the plan diagram. Our experimental results on 2D and 3D plan diagram query spaces, that are generated using about 80 query templates, show that this compact set of join order templates enable us to discover optimal plans of other queries in the query space whose cost is better than that of the optimizer plans.

Our experimental results show the utility of the proposed approach for both plan diagram query space and generic query space. For example, figure 1 (a) shows the distribution of plans for a 2D plan diagram query template with 6 relations on DS2 (described in section 2.1) when we took 32×32 grid. Two optimizer plans covered the entire query space of 1024 queries. The figure shows for each point on the grid, the optimal plan selected. The two plans that are denoted by △ and ■ in the figure, cover 784 and 240 queries respectively.

We used 5% of the queries (51 queries) for generating the plan logs and found two join tree templates that cover the optimizer plans of these plan logs generation queries. Figure 1 (b) shows the queries that are used for generating the logs (shown by the legend □). We then discovered the optimal plan of the remaining 973 queries using the join tree templates obtained. We discovered the same plan as the optimizer plan for 852 queries and for 121 queries (shown as ∗ in the figure) we discovered a different plan whose cost is lower than that of the optimizer plans. For these 121 queries, the plan selected by the optimizer is ■ (as shown in figure 1 (a) but the cost of the plan decreases when the plan △ is forced). The maximum decrease in the cost of the plan when compared to that the optimizer plan is 28.4, minimum being 0.091. The average decrease in the cost of the discovered plans is 10.128. Section 4.2 presents more results on plan diagram query space. Figure 2 shows the cost distribution of the plans where X and Y axes denote the grid points and the Z axis denotes the cost of the plan. The plan logs generation samples and the queries for which we discover a better plan are also marked in the figure.

Figure 1 (c) shows the refined plan diagram for the above query where we replaced the optimizer plan (i.e., ■) of the 121 queries with a better plan (△). Thus actual plan diagram of the query is different from the one that is obtained based on the optimizer plans.

Also as shown in tables 4 and 5 of section 4.3, for about 50 to 60% of the queries, our algorithm discovers the same as that of the optimizer plan and for about 12-20% of the queries, we discover a plan whose cost is less than that of the optimizer plan. For about 4-9% of the queries, the increase in the cost of the discovered plan is less than 10%. Thus, for majority of the queries, we discover plans whose cost either same or less than that of the optimizer plan. Section 4.5 discusses an example where the cost of the discovered plan is one-fifth of the optimizer plan.

Figure 3 shows the optimizer plan for a five relation query on the dataset DS5 (described in section 2.1). Figure 4 shows the plan selected by our approach, when the best plan was discovered based on the training join order templates. As can be observed, the cost of the discovered plan is roughly three fourth of the cost of the optimizer plan. Section 4.5 shows one more similar example.

1.1 Contributions

The main contributions of our work are as follows:

- We present an empirical analysis of the optimal plans and characterize them by means of a small set of join order templates.
Figure 2: X and Y axes denote the plan diagram grid points and Z axis denotes the cost of the plan. Cost Distribution of the 2D Plan Diagram Query Space on $32 \times 32$ grid. ‘.’ denotes the optimizer cost. □ shows the queries used for plan logs generation. * show the queries for which a better plan is discovered.

![Plan Diagram](image)

Figure 3: Optimizer Plan. Cost=378929. See table 7 for notation.

- Based on the observations we made about the plan space, we develop a plan discovery framework that discovers the optimal plan of a new query without enumerating the plan space.
- We implemented the plan discovery framework for a popular commercial DBMS and the utility of our framework on both plan diagram query space and generic query space, by means of empirical results is presented.

The organization of the paper is as follows. In section 2, we show the empirical results that lead us to make crucial observations about the optimal plan space. In section 3, we present our plan discovery framework. Section 4 reports the experimental results. Section 5 discusses the related work. And finally, section 6 concludes the paper with future work.
2 Understanding The Plan Space

In this section, we discuss the key observations about the plan space we made based on the experiments conducted.

2.1 Experimental Setup

We observed the nature of the plan space by studying the optimal plan space of queries executed on five different datasets DS1 to DS5. DS1 is based on the TPC-H benchmark database [11] and datasets DS2 to DS5 are four workloads created based on the TPC-DS benchmark database [10]. TPC-H and TPC-DS are decision support workloads with 8 and 24 relations respectively. Using the synthetic data generators provided with each of the benchmarks, we populated the five datasets. DS1 is a 1 GB TPC-H dataset and DS2, DS3, DS4 and DS5 are TPC-DS datasets with sizes 5 GB, 10 GB, 15 GB and 20 GB respectively. We populated these databases on a popular commercial database.

We then generated a random set of queries to be executed on these database and study their plan space. We wrote a program to generate queries by randomly selecting (i) the set of relations that are to be present in the query, (ii) the set of join conditions of the query, (iii) the set of attributes from the selected relations on which a select condition is to be imposed, (iv) the select predicate operator (=, >, <, ≥, etc), (v) the constant value for each selection predicate, (vi) the set of projected attributes and (vii) the set of aggregate attributes, if any.

2.2 Counting the plans

Execution plan of a query is represented by a binary tree, wherein the leaf nodes correspond to the relations of the query and the intermediate nodes correspond to the various relational operators like join, selection and projection. The cost of a query plan tree is mostly dominated by the cost of the join operations that are present in the query. Hence, given a query, finding its optimal query plan tree can be treated as finding its join tree that contains the order in which
the relations are joined and the join method that is to be employed for each join operation. For example, figures 5(a) and 5(b) denote the join trees of two different queries.

We define the term join order template of a join tree as the join tree in which the leaf nodes are replaced by any relation (wild card relation denoted by ‘[*]’). Thus, a join order template represents the structure of the join tree. Two join trees are defined to be isomorphic if their join order templates are the same. Figure 5(c) shows the join order template of the join trees shown in 5(a) and 5(b). A join order template compactly represents the cost of the join operations of the plan and hence is a good representation of the nature of the query plan.

From our experiments, we observed that the join trees of a large number of queries are isomorphic to each other. For example, the join trees shown in figure 5(a) and 5(b) are isomorphic to each other since they both have the same join order template shown in figure 5(c).

We conducted a large number of experiments to observe the distribution of the join order templates of a large number of queries on the five datasets described earlier.

2.3 Join order templates and query coverage

Experiment 1: We generated a large number of random queries, executed them on each of the datasets, obtained their optimizer plans and counted the query coverage of all the unique join order templates obtained. Query coverage of a join order template \( j \) is defined as the number of queries whose join order template is \( j \).

We conducted the experiment on the five datasets, varying the number of random queries executed between \( 10k \) and \( 100k \) in the increments of \( 10k \).

Table 1 lists the number of join order templates for each dataset for each size of the plan logs generation query sample. For a given dataset and a log generation query sample, if \( N_j \) and \( N_q \) denote the unique number of join order templates that cover the log generation query sample and the number of queries in the sample, the average coverage is defined as \( \frac{N_j}{N_q} \). As can be seen from the table, a small number of join order templates cover the optimizer plans of the dataset.

Also, by observing the column values of a particular dataset in the table, we can see that the number of new join order templates that get added when the sample dataset is increased from \( x \) queries to \( x+(10k) \) is small. And this number decreases drastically as the size of the sample dataset increases. For instance, for DS5, the number of new join order templates that get added when the size of the sample dataset increases from \( 10k \) to \( 20k \) is 50, whereas the increase is only 5 when the size of the sample dataset increases from \( 90k \) to \( 100k \). Thus, the set of join order
templates get saturated as the size of the sample dataset increases.

<table>
<thead>
<tr>
<th>No. of</th>
<th>DS2</th>
<th>DS3</th>
<th>DS4</th>
<th>DS5</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k</td>
<td>345</td>
<td>337</td>
<td>328</td>
<td>336</td>
</tr>
<tr>
<td>20k</td>
<td>396</td>
<td>393</td>
<td>393</td>
<td>386</td>
</tr>
<tr>
<td>30k</td>
<td>423</td>
<td>421</td>
<td>418</td>
<td>415</td>
</tr>
<tr>
<td>40k</td>
<td>441</td>
<td>441</td>
<td>442</td>
<td>436</td>
</tr>
<tr>
<td>50k</td>
<td>450</td>
<td>449</td>
<td>450</td>
<td>449</td>
</tr>
<tr>
<td>60k</td>
<td>463</td>
<td>466</td>
<td>464</td>
<td>464</td>
</tr>
<tr>
<td>70k</td>
<td>476</td>
<td>471</td>
<td>470</td>
<td>468</td>
</tr>
<tr>
<td>80k</td>
<td>481</td>
<td>474</td>
<td>476</td>
<td>473</td>
</tr>
<tr>
<td>90k</td>
<td>483</td>
<td>479</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>100k</td>
<td>488</td>
<td>485</td>
<td>486</td>
<td>485</td>
</tr>
</tbody>
</table>

Table 1: Number of Join Order Templates

In figure 6, we plot the coverage behavior of the join order templates for datasets DS2 and DS5 that are generated using 10k and 100k queries. X-axis show the percentage of the join order templates and for a given value of percentage of the join order templates, Y-axis show the percentage of the queries that are covered by them.

We can observe from the figure that for both the datasets with 10k queries, about 60% of the queries are covered by about 7% of the join order templates and about 20% of the join order templates cover 82% of the queries. As the number of queries increase, the average coverage of the join order templates increases drastically. For 100k queries, about 60% of the queries are covered by only 5% of the join order templates. Thus, a small percentage of the join order templates cover a huge percentage of the queries.

We studied how the join order templates collected after executing x queries, cover the next set of 10k queries. After the execution of every 10k queries, we noted the number of queries in the last 10k queries that are not covered by the join order templates collected till the previous iteration.
(we call it as *incremental coverage of the queries*). Figure 7 shows the number of new queries covered after execution of every $10^k$ queries for all the four datasets. We can observe that as the number of queries increase, the number of queries that are not covered by already collected set of join order templates decreases. Thus, the set of join order templates becomes saturated after execution of a huge number of queries.

Also, by comparing the values in table 1 and figure 7, we can observe that as the number of queries increase, the number of queries that are not already covered is of the order of the number of new join order template trees that get added. Thus, the coverage of the newly added join order templates is very less. Therefore, even if the log generation phase misses these join order templates, it cannot largely affect the accuracy of the plan discovery phase.

**Experiment 2:** We repeated the experiment by keeping the number of relations to be constant in each sample dataset. The number of relations for DS1 are 3 and 4 and for DS2 to DS5 the number of relations are 4, 6 and 8. The number of queries in each sample dataset are not fixed and are decided based on the following criterion. For each dataset, we keep generating queries with the required number of relations, execute them on the DBMS and get the join order templates. If the last unique join order template was obtained after executing $X$ number of queries, then we terminate the counting when $\alpha \times X$ number of queries were executed and no new unique join order template was obtained after $X$ queries are executed (where $\alpha$ is a small number)\(^3\). The percentage of the ratio of the number of join templates to the number of queries executed is very small for all the datasets. For DS1 this ratio is 0.324 and 0.00986 for 3 and 4 relation queries respectively. For DS2 to DS5, this ratio lies between 0.0041 and 0.117.

In figure 8, we plotted the behavior of the coverage of the join order templates. The set of join order templates is sorted in the decreasing order of their query coverage percentage, and an aggregate query coverage percentage computed according to the sorted order is plotted. X-axis shows the percentage of the join order templates and Y-axis shows the percentage of the aggregate query coverage percentage. A point $(x,y)$ on the plot shows that the top $x\%$ of the join

\[^3\]Though we used $\alpha=3$, we terminated the logs generation phase early in some cases due to lack of time.
order templates cover $y\%$ of the queries. We have shown the plots for DS1 and DS5. The plots behave similarly for DS2, DS3 and DS4.

Figure 8: Query Coverage of the JOT. See figure 6 for legend information.

As the number of relations in the dataset increases, the percentage of queries covered by top $t\%$ join order templates, for a specific value of $t$ increases. For instance, 5% of the join order templates covers approximately 34%, 60% and 75% of the queries for DS5 for 4, 6 and 8 relations respectively. Conversely, as the number of relations increases, the percentage of the top join order templates that cover a specific percentage of the queries decreases. For example, 80% of the queries are covered by top 25%, 10% and 6% join order templates for DS5 for 4, 6 and 8 relations respectively.

2.4 Cost Distribution of Optimal Plans

Experiment 3: We also observed the behavior of the cost distribution of the optimal plans. Given a dataset used in experiment 1, we select a random query $q$ from the dataset, and determine the set of join order templates $J$ that are compatible with $q$. We evaluate the cost of executing $q$ using each of the join order template in $J$, and we sort the members of $J$ in the decreasing order of the cost.

Figure 9 (best viewed in color) shows the cost distribution plot for 75 random queries generated on DS1. For each query in the set, the set of compatible join order templates are ordered in the increasing order of their forced cost (cost of executing the query using this particular join order template). We call such an ordering is called query specific ordering of the forced plans. The top 25% plans from this ordering is taken and the percentage of the cost difference with respect to the cost of the optimizer plan of the query are plotted on the Y-axis. Thus, X-axis shows the query specific ordering and Y-axis shows the percentage of the difference of the cost of the forced plan and the cost of the optimizer plan. Each curve shows the cost distribution of the forced plans. As we can observe from the figure, the plots of almost all the queries start at the zero value and the slope of the curve increases slowly for majority of the plots.

Based on the experiments 1, 2 and 3, we can derive the following observations:

---

*Those join order templates which can be potential join order templates of $q$.\]
Figure 9: Cost Distribution of Optimal Plans. X-axis: Query Specific Ordering of Forced Plans (ordered based on the cost of executing the query with the given plan). Y-axis: % of difference of cost of forced plan and the optimizer plan.

(a) The plan distribution is skewed: There are quite a few plans that are optimal for a large number of queries, and there are a large number of plans that are optimal for only a few queries.

(b) The optimal plan space of the queries can be represented by the set of join order templates of a sufficiently large number of queries.

(c) Given a query $q$, one or both of the following are true:

- There is high probability that the optimal plan of $q$ matches with the optimal plan of some query already optimized, and/or
- There are some optimal plans of other queries whose cost is near the cost of the optimal plan of $q$.

Based on the observations made, we devised a plan discovery framework that eliminates the plan enumeration phase.

### 3 Plan Discovery Framework

Figure 10 shows the block diagram of our plan discovery framework. During the plan logs generation phase, the query execution plans of the plan logs generation queries are collected from the DBMS. In an offline process, their join order templates are extracted and stored. The set of join order templates are updated as new queries are executed on the DBMS.

During the plan discovery phase, the optimal plan of a new query $q$ is determined as follows:

- $q$ is mapped to the compatible join order templates collected during the plan logs generation phase and a set of candidate plans for $q$ are generated.
- The cost of each of the candidate plans is evaluated and the least cost plan is selected for execution.
Figure 10: Plan Discovery Framework

Given a join order template \( j \), the set of candidate plans for \( q \) are generated as follows: Let \( G_j \) and \( G_q \) be the join graphs corresponding to the join conditions of \( j \) and \( q \) respectively. The isomorphic mapping of the nodes of the \( G_q \) to the nodes of \( G_j \) is formed and each such mapping gives one candidate plan. Though this problem is combinatorially complex, due to the join order constraints present in the query, many combinations get pruned, and our experimental results validate this.

In order to evaluate the cost of the candidate plans, the DBMS should allow the users to specify as input not only the query but also the plan that is to be used. The DBMS directly uses the plan that is specified by the user, instead of optimizing the query to find the optimal one.

Note that in order to measure the accuracy of the plan discovery, we take a set of queries for generation of the plan logs. But for production use, i.e., the actual deployment of the system, the set of workload queries act as plan logs generation queries. After generating the plan logs using sufficient number of queries, the plan discovery phase can be invoked.

The query coverage of the join order templates is skewed as shown in figure 6. Hence, instead of using the entire set of join order templates, the top \( t\% \) join order templates that cover \( c\% \) of the queries can be used where \( c \) can be say, 80% or 90%. This reduces the number of join order templates to be used for plan discovery without compromising the cost of the discovered plans for the majority of the queries since the omitted join order templates cover only a small percentage of the queries.

In the experimental results discussed in the next section, we show how the coverage affects the accuracy of the plan discovery. We also demonstrate that we can discover the query execution plan of a query by completely bypassing the plan enumeration phase and produce as good as and in some cases better than those selected by the DBMS query optimizer.
4 Experimental Results

4.1 Implementation Details

We implemented the plan discovery framework on a commercial database that has the capability of evaluating the cost of a query execution plan as discussed in section 3. We created five different databases based on DS1 to DS5 on five different systems. The details of the databases and their schema are discussed in section 2.1. We used two servers: server 1 to host the DBMS and server 2 to run our programs. We executed the plan log generation queries from server 2 onto the DBMS on server 1 and based on the optimizer plans that are returned by the DBMS, we computed the join order templates of the queries and stored them on server 2. We used the ODBTP [8] in our programs to connect to the DBMS. Server 1 has the Windows XP SP2 operating system (systems which hosted DS1,DS2 and DS3 has 1 GB RAM and those which hosted DS4 and DS5 has 2 GB RAM). Server 2 has the Fedora Core 10 operating system with 1 GB RAM. All our programs on server 2 are written in C++. We used MySQL database [9] to store the join order templates collected.

The DBMS outputs the optimal plan of a query in the form of an XML file. The users can evaluate the cost of executing a query using a specific query execution plan by submitting to the DBMS the query and the XML file corresponding to the plan. We parse the XML files using Xerces [12]. During the plan logs generation phase, the XML files of the optimal plans are parsed and the join order templates are extracted. During the plan discovery phase, the XML files corresponding to the candidate plans are generated to evaluate their costs. Generating the XML file corresponding to a query execution plan is a non-trivial task since there is no documentation available in order to figure out this mapping. We observed the optimal plans of a large number of queries and manually determined the mapping between various operators of the query execution plan and the XML file, using the DTD of the XML schema.

We divided the results section into four parts. In section 4.2, we present our results on the plan diagram query space. In section 4.3, we present the accuracy of our algorithm on generic query space. In section 4.4 discusses the effect of coverage on the accuracy of the plan discovery. Section 4.5 shows the plans that are missed by the optimizer.

4.2 Results on plan diagram query space

In this subsection, we present the effectiveness of our approach in a constrained scenario. During the plan logs generation phase of our algorithm, we execute a large number of queries of varying join conditions and selection predicates. Hence the query space covered is the entire possible space which is very large. We now consider a more constrained and specific query space, based on the the framework proposed in [20], which we call as plan diagram query space in this paper.

A query template is defined as a query for which one or more of the attributes from the relations of the query are selected as bind variables and a select predicate is defined on each of these bind variables whose selection value is defined at runtime. A set of queries can be generated based on a query template by considering all possible combinations of values that can be taken by the bind variables. For example shown below is a query template in which "ss_ext_wholesale_cost" and "wr_refunded_cash" are the bind variables.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DS2</td>
<td>5%</td>
<td>98.911</td>
<td>0.302</td>
<td>0.130</td>
<td>0.559</td>
<td>0.009</td>
<td>0.017</td>
<td>100.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>94.323</td>
<td>0.210</td>
<td>0.074</td>
<td>0.526</td>
<td>0.006</td>
<td>0.011</td>
<td>100.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>98.633</td>
<td>0.290</td>
<td>0.135</td>
<td>0.462</td>
<td>0.009</td>
<td>0.018</td>
<td>100.000</td>
<td>0.000</td>
</tr>
<tr>
<td>DS3</td>
<td>5%</td>
<td>96.588</td>
<td>0.084</td>
<td>0.005</td>
<td>0.240</td>
<td>0.008</td>
<td>0.008</td>
<td>100.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>90.052</td>
<td>0.610</td>
<td>0.032</td>
<td>1.562</td>
<td>0.086</td>
<td>0.112</td>
<td>100.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>90.820</td>
<td>0.608</td>
<td>0.036</td>
<td>1.562</td>
<td>0.093</td>
<td>0.131</td>
<td>100.000</td>
<td>0.000</td>
</tr>
<tr>
<td>DS4</td>
<td>5%</td>
<td>81.274</td>
<td>1.096</td>
<td>0.074</td>
<td>2.026</td>
<td>0.372</td>
<td>0.389</td>
<td>100.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>81.484</td>
<td>2.865</td>
<td>0.170</td>
<td>6.165</td>
<td>2.015</td>
<td>2.193</td>
<td>90.881</td>
<td>9.119</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>90.781</td>
<td>0.842</td>
<td>0.155</td>
<td>2.221</td>
<td>0.329</td>
<td>0.440</td>
<td>100.000</td>
<td>0.000</td>
</tr>
<tr>
<td>DS5</td>
<td>5%</td>
<td>87.359</td>
<td>2.788</td>
<td>0.000</td>
<td>5.546</td>
<td>1.803</td>
<td>1.890</td>
<td>95.121</td>
<td>4.879</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>99.833</td>
<td>0.054</td>
<td>0.029</td>
<td>0.078</td>
<td>0.000</td>
<td>0.015</td>
<td>100.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>88.047</td>
<td>0.746</td>
<td>-0.090</td>
<td>1.346</td>
<td>0.168</td>
<td>0.188</td>
<td>100.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 2: % Accuracy of the Plan Discovery on 2D Plan Diagram Query Space ( # Rel = 4)

```
SELECT sr.store_credit, ss.promo.sk, wr.returning.addr.sk
FROM store_sales, item, web_returns, store_returns
WHERE ss.item.sk = i.item.sk
AND wr.item.sk = i.item.sk
AND ss.ticket_number = sr.ticket_number
AND ss.ext_wholesale_cost < :1
AND wr.refunded_cash < :2
```

We generated about 80 different random query templates for our experiments. The number of join order templates that are covered the queries of these query templates lie between 1 and 20. The domain values taken by these two bind variables are divided into say 100 equal parts and for each of the values a query is generated by replacing their value in the above query template, thus giving rise to $100 \times 100$ queries.

[20] studies the plan space behavior for the plan diagram query space since the plan chosen is dependent on the selectivities of the relations that are involved in the query, which in turn are dependent on the select predicates applied on them. We use the plan diagram query space where in, we sample a subset of the queries in the query space, use them to generate the plan logs and collect the set of join order templates from them. For the rest of the points, we determine their plan based on the join order templates and compare the cost of the discovered plan with that of the optimizer plan. We thus verify the effectiveness of our approach in the scenarios where the query space is small and restricted.

We call query templates with two bind variables as 2D templates and in general, templates with $n$ bind variables as $n$D query templates. We generated random 2D and 3D query templates on DS2 to DS5 databases by varying the number of relations between 3 and 10 and tested the effectiveness of the plan discovery.

13
For 2D templates, we took the grid size to be $32 \times 32$. We varied the plan logs generation query sample size from 5% to 50% of the total number of queries and collected the join order templates. For the rest of the queries that were not part of the plan logs generation query sample, we determined their plan by invoking the plan discovery phase of our algorithm.

Table 2 shows the result for the above experiment when the number of relations are 4. For each dataset, the plan logs generation phase and the plan discovery phases are invoked five times and an average of statistics over these five runs are computed. The second column shows the percentage of the queries used for plan logs generation. The third column shows the average percentage of the queries for which the same plan as that of the optimizer plan is discovered. The next four columns show the average percentage error, average minimum error, average maximum error, minimum and maximum values of standard deviation of the error percentage values. The average percentage error is defined as the ratio of the sum of the error percentage for all the queries for which a different plan from the optimizer plan is selected to the total number of such queries. The next two columns show the percentage of the queries for which the error percentage is less than 5% and it is between 5 and 25% respectively.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DS2</td>
<td>5%</td>
<td>91.532</td>
<td>2.187</td>
<td>0.281</td>
<td>5.673</td>
<td>1.750</td>
<td>3.281</td>
<td>94.325</td>
<td>2.313</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>87.431</td>
<td>2.653</td>
<td>0.482</td>
<td>6.947</td>
<td>1.625</td>
<td>4.722</td>
<td>95.323</td>
<td>3.548</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>90.435</td>
<td>2.877</td>
<td>0.756</td>
<td>8.320</td>
<td>1.743</td>
<td>4.993</td>
<td>91.443</td>
<td>4.338</td>
</tr>
<tr>
<td>DS3</td>
<td>5%</td>
<td>82.346</td>
<td>2.358</td>
<td>1.353</td>
<td>7.440</td>
<td>1.548</td>
<td>6.531</td>
<td>93.256</td>
<td>3.858</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>80.442</td>
<td>2.890</td>
<td>1.877</td>
<td>7.932</td>
<td>1.248</td>
<td>4.231</td>
<td>90.253</td>
<td>5.384</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>78.432</td>
<td>3.653</td>
<td>1.232</td>
<td>8.448</td>
<td>1.873</td>
<td>5.471</td>
<td>94.364</td>
<td>3.850</td>
</tr>
<tr>
<td>DS4</td>
<td>5%</td>
<td>87.943</td>
<td>2.772</td>
<td>0.980</td>
<td>6.443</td>
<td>1.522</td>
<td>3.734</td>
<td>89.436</td>
<td>6.351</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>83.253</td>
<td>3.876</td>
<td>1.642</td>
<td>8.445</td>
<td>1.983</td>
<td>5.254</td>
<td>86.488</td>
<td>5.479</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>85.433</td>
<td>1.887</td>
<td>0.680</td>
<td>5.431</td>
<td>1.364</td>
<td>6.523</td>
<td>92.489</td>
<td>2.469</td>
</tr>
<tr>
<td>DS5</td>
<td>5%</td>
<td>81.834</td>
<td>4.332</td>
<td>1.240</td>
<td>7.442</td>
<td>1.890</td>
<td>4.279</td>
<td>85.432</td>
<td>11.334</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>85.363</td>
<td>2.552</td>
<td>2.754</td>
<td>8.455</td>
<td>1.082</td>
<td>6.479</td>
<td>89.352</td>
<td>8.429</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>79.337</td>
<td>2.335</td>
<td>1.748</td>
<td>7.435</td>
<td>1.539</td>
<td>5.362</td>
<td>82.494</td>
<td>14.365</td>
</tr>
</tbody>
</table>

Table 3: % Accuracy of the Plan Discovery on 3D Plan Diagram Query Space (# Rels = 6)

<table>
<thead>
<tr>
<th>DS</th>
<th>&lt; 0%</th>
<th>= 0%</th>
<th>≤ 0%</th>
<th>(0-10)%</th>
<th>(10-20)%</th>
<th>(20-30)%</th>
<th>(30-40)%</th>
<th>(40-50)%</th>
<th>&gt; 50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS2</td>
<td>14.610</td>
<td>72.663</td>
<td>87.273</td>
<td>3.704</td>
<td>0.882</td>
<td>0.441</td>
<td>2.910</td>
<td>2.205</td>
<td>2.586</td>
</tr>
<tr>
<td>DS4</td>
<td>17.023</td>
<td>60.188</td>
<td>77.211</td>
<td>7.418</td>
<td>3.380</td>
<td>0.939</td>
<td>4.225</td>
<td>3.568</td>
<td>3.258</td>
</tr>
</tbody>
</table>

Table 4: Distrib. of the Error % on Generic Query Space (# Rels = 4). (M, N] means > M and \( \leq N \)
It can be observed from the table that for most of the cases, the percentage of the queries for which it is same as that of the optimizer plan is discovered is between 87% to 96%. The lowest value of this percentage is 81% and highest is about 99%. Also the average error of the plan discovery lies in the range of 0.05 to 2.8. The standard deviation of the average error is also very less with about 2 as the maximum value. For majority of the case, almost for 100% of the queries for which a plan that is different from the optimizer plan is determined, the increase in the cost of the plan is less than 5%.

We repeated the experiment on a 3D plan diagram query space. We took $16 \times 16 \times 16$ to be the grid size. We generated random 3D query templates with 6 relations on the datasets DS2 to DS5. We varied the plan logs generation query sample from 5% to 50% and tested the accuracy of our plan discovery for the rest of the queries in the query space. For each value of size of the plan logs generation query sample, we divide the plan diagram queries into plan logs generation queries and evaluation queries five times and the average values of accuracy over the five runs of plan logs generation and plan discovery is determined.

Table 3 shows the result for the 3D plan diagram query space. We can see that percentage of the queries for which same plan is discovered is 80 to 90% and it decreases when compared to that of the 2D case. The average error of the plan discovery increased with maximum average error being 4.3. Also the different between the average minimum error and average maximum error increased when compared to the 2D case. The minimum standard deviation of the error values is between 1 and 2 and the maximum standard deviation lies in the range of 3 to 6. For about 80 to 95% of the queries, the increase in the cost of the discovered plan is less than 5%.

Based on the above two experiments, we can conclude that our approach works reasonably well in 2D and 3D plan diagram query spaces.

### 4.3 Results on generic query space

In this subsection, we present results obtained on DS2 to DS5 that show the accuracy of our plan discovery in a generic query space. We generated the plan logs from these datasets by varying the number of relations as 4, 6 and 8 on a huge number of queries. The plan logs generation phase is terminated by the $\alpha \times X$ condition (where $\alpha$ is an integer) described earlier in experiment 2 of section 2. The accuracy of the plan discovered is calculated by finding the percentage of error in plan discovery which is defined as $\frac{C_p - C_o}{C_o} \times 100$ where $C_p$ and $C_o$ denote the costs of the plans that are discovered and the optimizer plan respectively. The less the value of the error percentage, the better is the accuracy. A negative value of the error percentage shows that the cost of the plan discovered is less than that of the optimizer plan.

We tested the accuracy by running $20k$ test queries on each of the databases when the number of relations are 4, 6 and 8. Table 4 shows the percentage distribution of the error percentage values for the four datasets when there are 4 relations. For different values of the error percentage, we report the percentage of the test queries for which the error percentage takes that value. The columns of the tables represent the dataset, the percentage of the queries for which the error percentage is negative (queries for which a better than the optimizer plan is discovered), the percentage of queries for which the error percentage is 0% (queries for which same plan as the optimizer plan is discovered), the percentage of queries for which the error percentage is less
than or equal to 0%, between 0 and 10%, between 10 and 20%, between 20 and 30%, between 40 and 50%, and greater than 50%.

It can be seen that for about 14 to 20% of the queries, we discover a plan that is better than the optimizer plan. For about 50 to 72% of the queries, we discover the same plan as that of the optimizer plan. Also, for the remaining queries for which an expensive plan compared to that of the optimizer plan is discovered, for only 2 to 3% of these queries, the increase in the cost of the plan is more than 50%. Thus, the accuracy of the plan discovery is about 70 to 80%. And, the percentage of queries for which the discovered plan is highly expensive is very small.

Table 5 shows similar result when there are 6 and 8 relations. We only show the percentage of test queries for which the error percentage is less than 0% and 0%. The column which shows the percentage of queries for which the error percentage is less than or equal to 0%, shows the accuracy of the plan discovery which lies in the range of 70 to 80%.

Table 5: % Accuracy of the Plan Discovery on Generic Query Space

<table>
<thead>
<tr>
<th>DS</th>
<th>6 Rel</th>
<th>8 Rel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt; 0%</td>
<td>0%</td>
</tr>
<tr>
<td>DS2</td>
<td>12.53</td>
<td>69.72</td>
</tr>
<tr>
<td>DS3</td>
<td>18.94</td>
<td>61.79</td>
</tr>
<tr>
<td>DS4</td>
<td>16.65</td>
<td>58.93</td>
</tr>
<tr>
<td>DS5</td>
<td>17.52</td>
<td>53.32</td>
</tr>
</tbody>
</table>

4.4 Effect of coverage

We now show the effect of the coverage value on the accuracy of the plan discovery. We generation the plan logs on the DS1 dataset with the number of relations taken as 3, 4 and 5. We discovered the optimal plans of 1,000 queries with 3 relations after collecting the join order templates as described in experiment 2 of section 2. We experimented by taking the top t% join order templates that cover c% of the queries where c is varied from 100 to 10.

Figure 11 shows the error percentage of the plan discovery for different values of coverage. As can be observed from the figure, with 100% and 75% coverage, for about 2-4% of the queries, our plan discovery algorithm chooses a better plan, with the maximum decrease in the cost of the plan being 38%. For a large proportion of the queries (about 60-70%), the discovered plan cost is the same as that of the optimizer plan. For about 10% of the queries, the discovered plans are very expensive compared to the optimizer plans, with the increase in the cost to be in the range of 20% to 100%. The error percentage of the plan discovery, which is defined as the percentage of the ratio of the increase in the cost of the discovered plan to the cost of the optimizer plan, increases when the query coverage decreases to 50%. The cost of the discovered plan becomes more than that of the optimizer plan after 30-40% of the queries. The error percentage further increases with 25% and 10% coverage.

The average number of candidate plans whose cost is evaluated decreases as the coverage value decreases. With 100% coverage, the average number is 8.13. This number decreases to 3.97, 3.19, 3.12 and 2.96 when the coverage percentage decrease to 75%, 50%, 25% and 10% respectively.
Figure 11: Error % of the Plan Discovery (3 Rel) on DS1

Figure 12 shows the error percentage results of 4-relation and 5-relation datasets on DS1 when 1000 queries are used for testing. With 100% coverage, our discovered plan is the same as that of the optimizer plan for about 70% of the queries. Due to the absence of sufficient number of samples in the plan logs generation queries, the error percentage increases with 75% coverage. The average number of plans whose cost is evaluated are 17.16 and 7.32 for the 4-relation dataset with 100% and 75% coverage values, and 19.34 and 5 for the 5-relation dataset with 100% and 75% coverage values.

Figure 12: Error % of the Plan Discovery (4 & 5 Rel) on DS1

Table 6 shows for each coverage value (% Cov), the number of queries for which better plans were chosen (Better Plans), the number of queries for which same plan as the optimizer plan is discovered (Same Plans), the number of queries for which increase in the cost of the plan discovered is not more than 5% of the optimizer plan (< 5%), the number of queries for which the increase in the cost lies in the range of 5% and 20% (>= 5% and < 20%) and the average number of candidate plans whose cost is evaluated (Avg. # Plans). It can be observed from the table that as the coverage value decreases, the average number of plans whose cost is evaluated to discover a plan and the accuracy of the plan discovery decreases. But still the overall plan
discovery accuracy does not decrease drastically.

<table>
<thead>
<tr>
<th>% Cov</th>
<th>Better Plans</th>
<th>Same Plans</th>
<th>&lt; 5% Plans</th>
<th>&gt;= 5% &lt; 20% Plans</th>
<th>Avg. # Plans</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>41</td>
<td>416</td>
<td>362</td>
<td>157</td>
<td>8.13</td>
</tr>
<tr>
<td>95</td>
<td>41</td>
<td>387</td>
<td>353</td>
<td>142</td>
<td>7.19</td>
</tr>
<tr>
<td>90</td>
<td>41</td>
<td>382</td>
<td>356</td>
<td>135</td>
<td>6.79</td>
</tr>
<tr>
<td>85</td>
<td>29</td>
<td>371</td>
<td>379</td>
<td>141</td>
<td>6.50</td>
</tr>
<tr>
<td>80</td>
<td>27</td>
<td>369</td>
<td>382</td>
<td>137</td>
<td>6.12</td>
</tr>
<tr>
<td>75</td>
<td>21</td>
<td>357</td>
<td>391</td>
<td>151</td>
<td>3.97</td>
</tr>
</tbody>
</table>

Table 6: Accuracy of the Plan Discovery on DS1

Figure 13 shows the standard deviation of the costs of the top-3 plans that were chosen during the process of plan discovery for each query of 3 relations on DS1 where the standard deviation values are sorted in the increasing order. For all the coverage values, the standard deviation is less than 1.0 for about 40% of the queries. For about next 30% of the queries, the standard deviation is less than 10. This re-enforces our earlier observation that the set of candidate plans generated during the plan discovery phase not only contain the optimal plan but also other candidate plans differ from the optimal plan only by a small amount of cost and any of one of these plans can be used for execution.

Figure 13: SD of the cost of the Top-3 Plans on DS1

4.5 Plans missed by the Query Optimizer

Table 7(a) shows five relation query whose optimizer plan when executed on dataset DS3 is shown in table 7(b). After training the dataset on a training sample dataset, our algorithm discovered the plan for the query that is shown in table 7(c). The cost of the discovered plan is roughly five times less than that of the optimizer plan. The optimizer might have missed this plan due to errors in the cost estimation or statistics, or because of the early termination of the plan enumeration due to heuristics.
Figure 14: Plan Selected By Optimizer

It can be observed from the figure that the join order of the relations for plans (c), (d), (e) and (f) are one and the same. The relations I and SS are joined first, then their join result is joined subsequently with relations HD, C and CR in that order. One possible reason as to why the optimizer chose a different join order is that it might have failed to estimate correctly the size of the intermediate result of HD joining with the result of I joined with SS. Therefore, it would have pruned this particular join order at this stage, thus missing the plans.

We evaluated about 30 candidate plans to discover the best plan for the above query. Of these, 19 candidate plans have cost less than that of the optimizer plan. Three best plans apart from the discovered plan are shown in table 7(d), 7(e) and 7(f) whose costs are 138257, 139522 and 141146 respectively. The costs of the next three best candidate plans are 160445, 161883 and 163101. This provides strong evidence that the cost values of many candidate plans are close to that of the optimal plan and using any one of them to discover the plan would result in a “good” discovered plan. Also, due to this observation we can conclude that even if the training queries do not cover the entire set of join order templates, “bad” plans will not be discovered.

4.6 Discussion

Our experiments prove that our approach works in a constraint query space like 2D and 3D plan diagram query spaces. We also show that our accuracy is about 70 to 80% even for a generic query space. In this work, we do not provide any bounds on costs of the plan discovered. Since we deal with a generic plan space, it is very difficult to estimate the cost of the optimal plan (plan with the least cost) and run a check as to how close is the cost of the discovered plan with respect to that. However, such an analysis should be possible in a constraint query space, where the parameters that change across the queries are fixed and all the queries belong to a query template.

To train our system, we randomly sample the query space and extract the join order templates. Our terminating condition runs a large number of queries in order to make sure that no new join order templates can be found. If any assumptions can be made about the distribution of the optimizer plans in the query space then the sampling can be done with a smaller number of training queries. The assumptions made will help in determining the join order templates of a
SELECT cr_returning_cdemo_sk, 
hd_buy_potential, 
cr_net_loss, cr_return_amount, 
ss_store_sk, ss_ext_tax, 
hd_income_band_sk, 
ss_net_paid_inc_tax, 
ss_wholesale_cost 
FROM item, store_sales, 
household_demographics, 
customer, catalog_returns 
WHERE ss_item_sk > 720 
AND hd_demo_sk >= 288 
AND ss_item_sk = i_item_sk 
AND ss_hdemo_sk = hd_demo_sk 
AND c_current_hdemo_sk = hd_demo_sk 
AND cr_returning_hdemo_sk = hd_demo_sk 

(a) Query

(b) Optimizer Plan

(c) Discovered Plan

(d) Better Plan

(e) Better Plan

(f) Better Plan

Table 7: Plans Missed By the Optimizer. Non-Leaf Nodes Notation: HJ - hash join, MJ - merge join, CIS - clustered index scan, SRT - sort. Leaf Nodes Notation: I - item, SS - store_sales, C - customer, CR - catalog_returns, HD - household_demographics. Cost of each node is shown next to it. Cost of the plans - (b) 633595, (c) 136888, (d) 138257, (e) 139522, (f) 141146.

large area of query space, by observing the join order templates of a small number of training samples. For instance, we can assume that the region of optimality of a query plan is either convex or convex polytopes (e.g. [13, 14]). Using this property, once the queries on the boundary of this convex region are sampled, the queries that lie inside the convex region need not be sampled since their join order templates can be estimated. But, since we apply our algorithm on
a generic query space, no such assumptions can be made.

Our current system assumes that the query optimizer of the DBMS is a blackbox and we do not have access to the internals of it. A more efficient plan discovery can be designed if we can embed the plan discovery module directly into the query optimizer. Since the plan discovery module will have access to all the cost estimates and the intermediary decisions that will be taken by the optimizer in the process of query optimization, more information can be collected and stored in addition to the join order templates during the training phase.

Although the main aim of our system is to by-pass the plan enumeration to determine the optimal plan, it can also be used by the database system builders to test the effectiveness of the cost model and the cost estimates of the query optimizer. For each of the cases for which our system discovers a plan that is better than the optimizer plan, the root cause for this behavior can be determined by backtracking the process of query optimization. This will help in identifying the potential errors in the cost model and the statistics. We are currently implementing a variation of our system which focuses on testing the “goodness” of the optimizer.

5 Related Work

Since the problem of query optimization (QO) is a very hard, many heuristic techniques have been proposed to reduce the complexity of the optimization. Different heuristics are developed based on simulated annealing (SA) [17], iterative improvement (II) and two phase optimization [25, 24, 15], genetic algorithms [24, 15] and iterative dynamic programming [18]. [19] studies the trade-off between QO and execution cost in DBS3 parallel database system when various heuristics like dynamic programming, SA and II are employed. [22] studies various deterministic and heuristic join ordering algorithms and compares them empirically. We do not employ any of these heuristics in our approach.

[3] proposes elimination of the plan space enumeration by selecting a set of random valid candidate plans from the search space and choosing the least cost candidate plan as the optimal plan. The work discusses how to generated uniformly distributed random query plans. On the other hand, in our work we do not use any random plans to determine the plan for a new query. Instead, we use only those plans which are optimal at some point in the query space since during our plan logs generation phase we collect the join order templates of only optimal plans.

LEO [23] is a framework that learns from the mistakes incurred by the query optimizer during the calculation of the statistics and cardinality estimates, to reduce the errors that might occur due to incorrect statistics. Based on the actual executions of the query, LEO validates the optimizer’s model incrementally to determine which parts of the model are erroneous and adjusts the model accordingly. In our work, we learn the nature of the optimal plans as compared to the learning of statistics by LEO.

Two works that determine the optimal plan of a query bypassing the plan enumeration are parametric query optimization (PQO) and PLASTIC. PQO [16, 1, 4, 13, 14] attempts to identify several execution plans, each of which is optimal for a subset of all possible values of the run-time parameters. Determining the optimal plan for a query can be done without invoking the query optimizer and by simply finding the execution plan that corresponds to the run-time parameters at that instance. But precomputing the optimal
plans at all possible parameter values is very expensive. Unlike PPO, we do not determine the optimal plans of all possible queries or query regions. Moreover, the join order templates we collect, can be a by-product of the execution of the normal workload queries and hence we do not need to specially fire queries to get the optimal plans.

PLASTIC [5] clusters the queries into groups based on the query features. For each query cluster, a query plan that is selected by the optimizer is stored. If the number of query clusters is sufficiently large, given a new query, a classification technique is used to determine which query cluster the query belongs to. Our work differs from PLASTIC where we use join order templates as the representative features of the optimal plan space. We do not explicitly cluster the queries based on their features, but the join order templates naturally group all the queries that share common properties in terms of their optimal plans.

Our work differs from the earlier work as follows: (i) The plan logs generation phase of our framework which we invoke before the discovery of the optimal plan can be an entirely offline process; and (ii) Our framework can be implemented by considering the DBMS (and the query optimization) as a black box, without modifying into the internals of the system.

[20] analyzes the plan space using plan diagrams as discussed earlier. [6] addresses the problem of reducing the number of plans in the plan diagram, without affecting the query processing quality. The reduction problem is proved to be NP-hard and a greedy based algorithm is proposed. The aim of the work is similar to that of our paper: can we identify a small set of plans that can be used to obtain the optimal plan of all the queries in the query space. But unlike their work, we do not precompute the plan diagram by executing queries at all the points in the query space. We only sample a set of queries, determine their optimal plans and extract the join order templates. Our experimental results show that our approach gives good results in plan diagram query space. Also, our method is not limited to plan diagram query space and works in a generic query space also.

6 Conclusions And Future Work

We built a system that determines the optimal plan of a query without enumerating the plan space. We extracted the join order templates of a large number of queries and used them to determine the optimal plan of new queries with good accuracy. We evaluated the utility of the system by means of empirical results. This work is a first step in building a learning based query optimizer to make the plan discovery phase more accurate and scalable.

Our ongoing work includes:

- We are currently working on a thorough evaluation of the system on a wide variety of databases with varying number of relations, varying sizes of databases and different workloads.
- We have not used any special data structure to store the join order templates collected nor we have used any fast algorithms for finding the matching join order templates during the plan discovery phase phase. We believe that both these aspects can be done more efficiently.

But for the evaluation of our proposed framework, we fired queries to get the optimal plans.
In this paper, we did not differentiate the queries with aggregate functions and ‘group bys’ from the normal SPJ queries during plan logs generation or plan discovery phases. We are currently looking at the optimal plans of such queries to see if any changes need to be made to the plan logs generation or plan discovery phases.

If an approximate cost model of the query optimizer can be derived, the invocation of the optimizer to estimate the cost of the plan can be avoided. This makes the plan discovery phase more efficient.

References


