DivIDE: Efficient Diversification for Interactive Data Exploration

Hina A. Khan
School of ITEE
The University of Queensland
Queensland, Australia
h.khan3@uq.edu.au

Mohamed A. Sharaf
School of ITEE
The University of Queensland
Queensland, Australia
m.sharaf@uq.edu.au

Abdullah Albarrak
School of ITEE
The University of Queensland
Queensland, Australia
a.albarrak@uq.edu.au

ABSTRACT

Today, Interactive Data Exploration (IDE) has become a main constituent of many discovery-oriented applications, in which users repeatedly submit exploratory queries to identify interesting subspaces in large data sets. Returning relevant yet diverse results to such queries provides users with quick insights into a rather large data space. Meanwhile, search results diversification adds additional cost to an already computationally expensive exploration process. To address this challenge, in this paper, we propose a novel diversification scheme called *DivIDE*, which targets the problem of efficiently diversifying the results of queries posed during data exploration sessions. In particular, our scheme exploits the properties of data diversification functions while leveraging the natural overlap occurring between the results of different queries so that to provide significant reductions in processing costs. Our extensive experimental evaluation on both synthetic and real data sets shows the significant benefits provided by our scheme as compared to existing methods.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing*

General Terms

Algorithms, Design, Experimentation, Performance.

1. INTRODUCTION

Interactive Data Exploration (IDE) platforms assist users to discover interesting objects within large volumes of data [5]. This is an essential step in a widely diverse set of discovery-oriented applications in both science and business, where IDE allows identifying manageable portions of data for further analysis using machine learning and data mining models [31]. In those applications, users try to make sense of the underlying data space by iteratively executing numerous exploratory queries [10].

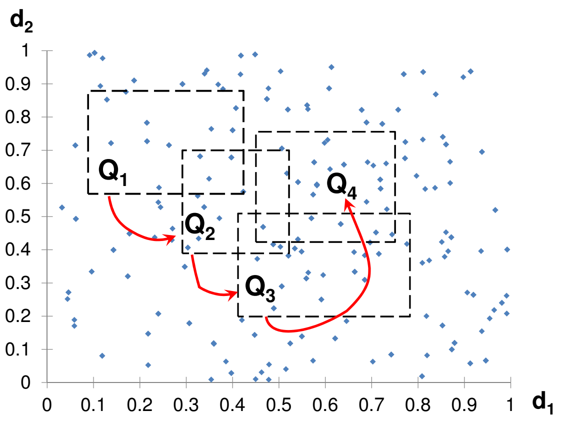

Figure 1: Series of range queries generated in a user session

IDE platforms rely on several techniques that can be broadly classified as: 1) *query refinement*, and 2) *representative data extraction*. The goal of query refinement techniques is to assist users to quickly navigate and find interesting regions in a large data space (e.g., [35, 23, 34, 6, 3]). Orthogonally, representative data extraction aims to provide users with a concise and meaningful representation of the data in each of those explored regions. In particular, extracting a few tuples from a query result to provide quick insights in the potentially huge answer space. Beyond the well-studied *top-k* (e.g., [18, 17]) and *skyline* (e.g., [33, 30]) operators, *diversification* is rapidly becoming the technique of choice for data summarization in IDE platforms (e.g., [12, 13, 14, 36, 1, 21, 20]).

For instance, consider the Sloan Digital Sky Survey (SDSS) science database[1], which describes over 140 million objects and is over 30 TB in size. SDSS provides several levels of query interface to the public via the SkyServer website. In SDSS, and other data exploration platforms, users typically interact with the platform with some initial hypothesis in mind and start firing queries hoping they might retrieve results that support their hypothesis. Providing diverse and summarized subsets of those query results helps users to

[1]http://www.sdss.org

quickly asses the relevance of the retrieved data to their initial hypothesis [19, 5].

To highlight the interplay between query navigation and data diversification, consider the following example based on the SDSS database:

Example 1. *Assume a casual user, who is curious about finding a region in sky with stars showing some interesting color features. However, the user has no prior information about the location of such regions. The user will then access SDSS via its web interface and she will start querying the database with some intuitive coordinate values for specifying a certain area of the sky. As shown in Figure 1 the user will first pose query Q_1 and inspects the summarized results provided through diversification. If the results are unsatisfactory, she might pose another query Q_2, which she formulates manually or using some automated query navigation technique. This iterative process continues until the user finally finds the area she is looking for, which is represented by query Q_4 in Figure 1.*

Notice that in the example above, the user interaction with the database takes the form of an *exploratory session*, or session for short [5]. In a session, a user issues a sequence of related queries, in which each query serves as a springboard to the next. A session typically involves exploring the data space by a lengthy sequence of queries, which consumes non-trivial amount of computing resources. While diversification, like other data summarization techniques, provides users with quick insights into the query answers, it adds additional complexity and requires extra computational resources. Meanwhile, delivering near realtime performance remains an essential requirement for IDE platforms so that to match the intrinsic nature of interactive and iterative data exploration to ensure user satisfaction.

Motivated by the need to efficiently provide users with effective insights during data exploration, in this paper we propose the *Diversification for Interactive Data Exploration (DivIDE)* scheme. DivIDE targets the problem of efficiently diversifying the results of multiple queries within and across different exploratory sessions. Specifically, our goal is to develop methods that are able to strike a fine balance between the efficiency of diversification (i.e., processing cost) and its effectiveness (i.e., accuracy of diversification) during data exploration. In our previous work, we have investigated the orthogonal problem of diversifying the results of simultaneous queries that are executed in a batch mode. Towards that, we have proposed the *DoS* scheme, which utilizes the overlaps between query results to eliminate redundant computations.

DivIDE also leverages the fact that data exploration sessions involve a sequence of overlapping queries and results. However, that overlap occurs over time and is not known a prior. Thus, to utilize that overlap, DivIDE partitions the query results into new results and historical overlapping results. A model-based diversification method is then employed for effectively locating the diverse results from both partitions.

The main contributions of this paper are summarized as follows:

- We formulate the problem of result diversification for interactive data exploration, together with metrics that captures both the efficiency and effectiveness of diversification (**Section 2**).
- We adapt schemes that leverage the overlap between query results within and across exploratory sessions towards efficient diversification (**Section 3.2**).
- We propose the novel DivIDE scheme, which utilizes a model-based approach that is particularly suitable for the efficient and effective diversification in IDE (**Section 3.3**).
- We conduct extensive experimental evaluation on real and synthetic data sets, which compare the performance of multiple diversification schemes and illustrate the benefits achieved by DivIDE (**Section 5**).

Roadmap: We define the search result diversification for interactive data exploration in Section 2. Next we introduce our DivIDE scheme in section 3. The evaluation testbed and results are reported in Section 4 and Section 5 respectively. We conclude in Section 6.

2. BACKGROUND AND RELATED WORK

Scientists and analysts typically explore a relational database through a sequence of exploratory queries to discover interesting insights or verify a particular hypothesis. An *exploratory query* is typically based on range predicates represented as a multi-dimensional box or hyper-rectangle. Such queries retrieve a number of tuples, or objects, from the database, which can be generally viewed as a set of attribute values represented as data points in a multi-dimensional data space and can be further grouped and aggregated.

In order to guide the user throughout the data space, a data exploration platform is expected to manipulate the query result to automatically show views that could assist the analyst to further explore related queries. This process starts as a user submits an *initial* exploratory query and continues until one or more *target* queries have been identified, where each target query basically defines the contour lines of a portion of the data space that contain data of high interest to the user. As a consequence, the series of queries submitted by a user during an exploratory session are typically correlated in a sense that the user formulates the next query in the sequence after having reviewed the results of previous queries [7]. This leads to an iterative process, in which a user executes numerous selection queries iteratively using different predicates [10]. For instance, reconsider example 1 where a user is trying to find a region of stars with some interesting color features. The sequence of queries illustrated in Figure 1 starts with query Q_1 with predicates: Q_1= `0.750 <` dim_1 `< 0.425 AND 0.575 <` dim_2 `< 0.882` and ends with query Q_4 with predicates Q_4= `0.450 <` dim_1 `< 0.750 AND 0.430 <` dim_2 `< 0.750`.

Clearly, the exploratory process illustrated above is ad-hoc and consumes a lot of resources during the execution of the

Table 1: Table of Symbols.

Symbol	Description
N	Total number of queries across all sessions
k	Diverse subset size
Q_c	Current query to be diversified
X_c	Result set of Q_c
S_c	Diverse subset of X_c
$\mathbb{Q}_{\mathbb{H}}$	Set of history queries
$\mathbb{S}_{\mathbb{H}}$	Union of diverse subsets of history queries
$Q_{O,c}$	Set of queries overlapping with Q_c
$S_{O,c}$	Union of diverse subsets overlapping with X_c
$S_{R,c}$	Reusable diverse results for Q_c
θ	Deviation Threshold

different queries fired during this process [26]. This has motivated a body of research that aims to leverage the natural *overlap* that occurs between the *raw* data accessed by such queries towards efficient and optimized overall query processing (e.g., [5, 28]).

Moreover, since exploratory queries are imprecise by nature and potentially return large result sets, it is essential to apply some post-processing techniques to summarize this data. Otherwise, some important insights are left hidden under a pile of raw data records. Extracting representative tuples from a query result that provide quick insights into those results is an important functionality in data exploration. That motivated the need for developing effective methods to assist users in quickly locating results of high importance. Example of such methods include the well-studied *top-k* and *skyline* queries [33]. While both queries provide significant advantages in preference and personalized databases, their application is limited in the absence of preference parameters, as is the case in data exploration [29]. This motivates the need for novel methods for extracting meaningful representative tuples from a query answer (e.g., [36, 24, 2]), including: *diversification*, *regret minimization*, and *automated ranking*. In this work, we focus on diversification as the method of choice for representative data extraction, as explained next.

2.1 Search Result Diversification

It is essential for an IDE platform to help the user make quick decisions on the basis of few *representative* results, instead of sifting through all the results looking for those interesting insights. Search results diversification lends itself as an effective solution for extracting those representative results. Formally defined, let $X = \{x_1, \ldots, x_m\}$ be a set of results generated in response to some user query Q. In general, the goal of result diversification is to select a subset S^* of X with $|S^*| = k$, $k \leq m$, such that the diversity of the results in S^* is maximized.

There are various definitions of diversity [36]. Most of them can be classified in one of the following categories: (i) content based, i.e., selecting results that are dissimilar to each other (e.g., [36]) (ii) novelty based, i.e., selecting results that contain new information when compared to what was previously presented to the user (e.g., [8]) and (iii) semantic coverage based, i.e., selecting results that belong to different categories or topics (e.g., [1]).

In this work, we primarily focus on the widely used content-based definition of diversity. Content diversity is an instance of the *p-dispersion* problem [16], whose objective is to maximize the overall dissimilarity within a set of selected objects.

In particular, given a metric d that measures the distance between two results, e.g., the Euclidean distance among two data points, the diversity of a set S is measured by a *diversity function* $f(S, d)$ that captures the dissimilarity between the results in S. To that end, a number of different diversity functions have been employed in the literature, among which previous research has mostly focused on measuring diversity based on either the *average* or the *minimum* of the pairwise distances between results [12, 36]. We focus on the first of those variants (i.e., average) known as max-sum diversity measure, as it adopts a more balanced view that considers all the results in S, defined as:

$$f(S,d) = \frac{1}{k(k-1)} \sum_{i=1}^{k} \sum_{j>i}^{k} d(x_i, x_j)$$

Putting it together, the max-sum diversification problem is defined as follows:

Definition 1. *Let X be the set of results that satisfy a user query Q and k be a positive integer such that $k \leq |X|$. Let also d be a distance metric and f a diversity function. Then, max-sum Diversification is defined as selecting a subset S^* of X, such that:*

$$S^* = \underset{\substack{S \subseteq X \\ |S|=k}}{\operatorname{argmax}} f(S,d)$$

In general, search results diversification problem has been shown to be NP-hard(e.g.,[15]), which essentially renders the location of an optimally diverse set S^* impractical, especially for large data sets. Among many approximation methods used to solve this problem, the greedy heuristics are the simplest and perhaps the most widely used methods. These greedy algorithms have been proven to provide $1/2$-approximations to the optimal solution ([27]).

The Greedy heuristic initializes the diversified set S by first selecting a random result in X. Then, it proceeds with a number of iterations, until k results have been selected. At each iteration, the result with the maximum sum of distance from the already selected results is added to the partially computed diverse set S. Clearly, the complexity of this Greedy heuristic in terms of computed pairwise distances is $O(|k|^2|X|)$. However if the distance computations across different iterations of greedy are cached this leads to further reduction in number of pairwise distance calculation and the complexity of greedy is considered to be $O(|k||X|)$ ([22]), which makes it a very reasonable choice in interactive online search environments. (The Greedy heuristic is shown in Algorithm 1.)

2.2 Problem Definition

Like other representative data extraction techniques, applying diversification adds additional complexity and requires extra computational resources. For instance, consider a simple greedy heuristic, as presented above, which provides

Algorithm 1 Greedy.

Input: A set of query results X, an integer k.
Output: A set S^k with the k most diverse results in X.

```
1: $x \leftarrow$ random result $\in X$
2: $i \leftarrow 1$
3: $S^i \leftarrow \{x\}$
4: $i \leftarrow i + 1$
5: while $i < k$ do
6:     $x_{max} \leftarrow \text{argmax}_{x_{max} \in X} \sum_{x_j \in S^{i-1}} d(x_{max}, x_j)$
7:     $S^i \leftarrow S^{i-1} \cup \{x_{max}\}$
8:     $i \leftarrow i + 1$
9: end while
10: return $S^k$
```

a reasonable approximation for the diversification problem within a polynomial time. In an IDE platform where multiple user sessions are supported even a polynomial time diversification algorithm is bound to take the query processing time beyond acceptable limits. In particular, it is expected for an IDE platform to deliver a near real time performance that is suitable for the intrinsic nature of iterative and interactive data exploration, which is typically captured by means of the *time-to-insight (TTI)* measures. Notice that TTI depends on many factors including: the complexity of the exploration, the efficiency in processing the different exploratory tasks, and the effectiveness in navigating the query space.

In this work, we focus on the particular component of TTI that pertains to the post-processing of results to extract representative data by means of diversification. Specifically, consider an IDE setting, where a series of N related queries are submitted by a number of different users. Further, let Q_i be an i^{th} user query and let X_i be the set of results that satisfy Q_i and k_i be a positive integer with $k_i \leq |X_i|$. Let also d be a distance metric and f a diversity function. Accordingly, we define the following metrics:

Definition 2. *Diversification Cost, $C(S_i)$, is defined as the processing cost, in terms of number of distance and comparison operations, required to process a result X_i to produce a diversified result S_i of size k. Accordingly, the average diversification cost for N queries is: $\frac{1}{N}\sum_{i=1}^{N} C(S_i)$.*

Definition 3. *Diversification Quality, $D(S_i)$, is defined as the value of diversity $f(S_i, d)$ offered by the diversified result S_i. Accordingly, the average diversification quality for N queries is: $\frac{1}{N}\sum_{i=1}^{N} D(S_i)$.*

Notice that the results of those queries are expected to exhibit some degree of overlap. For instance, the series of queries illustrated in Figure 1 are expected to access overlapping portions of the raw data space as well as to retrieve overlapping results. Such overlap in results is expected to take place during any typical exploratory session as well as across different sessions intimated by different users. In this work, we leverage that overlap between query results in order to achieve high efficiency in presenting a user with representatives that are generated by means of diversification.

Algorithm 2 SGC.

Input: Query Result set $X_c = (x_1, x_2...x_m)$, an integer k, Overlapping diverse results from previous query $S_{R,c}$
Output: Set S^k=(s_1, s_2...s_k) with the k most diverse items of X_c.

```
1: $i \leftarrow |S_{R,c}|$
2: $S^i \leftarrow S^i \cup S_{R,c}$
3: $i \leftarrow i + 1$
4: while $i < k$ do
5:     $x_{max} \leftarrow argmax_{x_{max} \in X_c} \sum_{x_j \in S^{i-1}} d(x_{max}, x_j))$
6:     $S^i \leftarrow S^{i-1} \cup \{x_{max}\}$
7:     $i \leftarrow i + 1$
8: end while
9: return $S^k$
```

Specifically, our goal is to strike a fine balance between the efficiency of diversification (i.e., minimize $\frac{1}{N}\sum_{i=1}^{N} C(S_i)$) and its effectiveness (i.e., maximize $\frac{1}{N}\sum_{i=1}^{N} D(S_i)$). In the next section, we present our schemes and discuss their impact on both the efficiency and effectiveness of diversification.

3. INTERACTIVE DIVERSIFICATION

In this section, we present our *DivIDE* scheme for the efficient diversification in interactive data exploration platforms. The main idea underlying DivIDE is to leverage the overlap in a sequence of queries (such as the one shown in Figure 1) in order to minimize the computational cost incurred in diversifying each query result set. Before going through the details of DivIDE, we first present two baseline solutions to the problem of diversification in IDE, namely: Greedy Construction, and Stream Greedy Construction (SGC).

3.1 Greedy Construction

In this baseline solution, the *Greedy Construction* algorithm (or *Greedy* for short), is directly applied for the diversification of results in IDE. In particular, Greedy (as presented in algorithm 1) processes each query individually without taking into consideration the overlapping in results between different queries.

Recall from Section 2.1, the computational complexity of diversifying a single query using Greedy algorithm is $O(k|X|)$, where $|X|$ is the size of the query result and k is the size of the diverse subset. Clearly, applying Greedy algorithm independently to each IDE query, results in a computational complexity that increases linearly with the increase in number of total queries across all user sessions (i.e., N). That is, the complexity of Greedy algorithm for N queries is simply computed as: $O(k|X_1|) + \ldots + O(k|X_N)|)$.

3.2 Stream Greedy Construction (SGC)

Stream Greedy Construction (SGC) [11] extends the basic Greedy scheme for the case of continuous data streams. In particular, it perceives diversification as a continuous query, in which the k most diverse results need to be evaluated for each sliding window over the data stream. Clearly, as the window slides over the data stream, some new data is added and some expire, leaving some significant overlap between any two consecutive windows w_t and w_{t+1}. The premise underlying SGC is that instead of re-evaluating all the k diverse results for each sliding window, SGC initializes the diverse subset of the current window w_t using the diverse results from the previous window w_{t-1} that are still valid (i.e., unexpired). Let the number of such overlapping results be r, then SGC only computes $k-r$ diverse results for the current sliding window. SGC has been shown to be an effective and efficient heuristic for diversifying data streams.

In this work, we also consider SGC as a second baseline for the diversification of a series of related queries within a single user session. Specifically, as shown in figure 1, each exploratory query can be perceived as a sliding window over the data space. Since different queries within an exploratory session typically explore the data space in a close vicinity to each other, it is very likely for two consecutive queries Q_i and Q_{i+1} to have common results, similar to two consecutive sliding windows w_t and w_{t+1} in a data stream. That is, sliding a window over a dynamic data stream resembles moving a query over a static database.

To clarify the intuition above, consider queries Q_1 and Q_2 in Figure 1. Clearly, their result sets X_1 and X_2, have some shared results that can be represented as $X_1 \cap X_2$. Let also S_1 and S_2 be the sets of diverse results for X_1 and X_2, respectively. If any diverse result in S_2 comes from $X_1 \cap X_2$, then there are chances that it can be part of S_1 as well. Therefore, in order to save distance computations while diversifying X_2, the overlapping diverse results from S_1 are retrieved (i.e., $X_2 \cap S_1$), and are used to initialize the diverse subset S_2 of Q_2. Typically, the number of results in $X_2 \cap S_1$ (i.e., r) is less than k, hence, only $k-r$ additional diverse results are computed to generate the complete diverse set S_2. Obviously, reusing some of the previously computed diverse results, allows SGC to execute less iterations than the greedy heuristic described in Section 2.1 above. That reduction in the number of iterations, however depends on the number of overlapping diverse results obtained from the previous query (i.e., $|X_2 \cap S_1|$).

3.3 Diversification for IDE (DivIDE)

While the SGC heuristic reduces the cost of diversification in terms of number of greedy iterations needed for selecting k diverse results from a result set X_i, it still relies solely on the overlap that occurs only between the results of two consecutive queries in a session (i.e., $X_i \cap X_{i-1}$). In contrast to data streams, in IDE platforms, queries are generated in any order and later queries in a session might still have some significant overlap with some of the earlier queries in that same session. Meanwhile, SGC as described above, is oblivious to that kind of overlap except for the immediate overlap between two consecutive queries within a session. Additionally, in a more general setting where there are multiple sessions activated by multiple users, overlapping between query results is naturally expected to occur across those different sessions. For example two users exploring roughly the same data subspace will generate queries that are quite similar to each other, however the order of issuing those queries may vary depending upon the exploration steps conducted by each user.

To formally express that overlap between exploratory queries, consider a current query Q_c for which a diversified set S_c is to be computed, and a history of processed queries $\mathbb{Q}_H = \{Q_1, Q_2, ..., Q_{c-1}\}$. Hence, there exists $\mathbb{Q}_{O,c} \subseteq \mathbb{Q}_H$ such that the result X_i of each query $Q_i \in \mathbb{Q}_{O,c}$ overlaps with the result X_c of the current query Q_c (i.e., $X_i \cap X_c \neq \phi$).

Clearly, the diversified results of these overlapping queries $\mathbb{Q}_{O,c}$ can be utilized for reducing the cost of diversifying Q_c. Particularly, in this work, we propose using a *cache* of diversified results for improving the efficiency of diversification. A Cache $\mathbb{S}_H$, which contains the diversified results of all previously processed queries $\mathbb{Q}_H$, is expressed as follows:

DEFINITION 4. *Cached Diverse Results, $\mathbb{S}_H$, is the set of diversified results corresponding to the queries in $\mathbb{Q}_H$. That is, $\mathbb{S}_H = \{S_1, S_2, ..., S_{c-1}\}$.*

Given a query Q_c, it is then straightforward to fetch the cached diverse results of the set of queries $\mathbb{Q}_{O,c}$ that overlap with Q_c. We denote the cached diverse results of queries overlapping with Q_c as $S_{O,c}$, which is defined as:

DEFINITION 5. *Diverse Results of overlapping Queries, $S_{O,c}$, is the union of the diversified sets of all the queries in $\mathbb{Q}_{O,c}$.*

Intuitively, $S_{O,c}$ is expected to contain some points that are common with the results of query Q_c (i.e., $X_c \cap S_{O,c} \neq \phi$). That set of common points $S_{R,c}$ provides an excellent opportunity to *reuse* in constructing the diverse set S_c and is simply defined as follows:

DEFINITION 6. *Reusable Diverse Results, $S_{R,c}$, such that $S_{R,c} \subseteq S_{O,c}$ and contains all diverse results in $S_{O,c}$ that fall in the range of Q_c (i.e., $S_{O,c} \cap X_c$).*

One idea towards utilizing $S_{R,c}$ when diversifying Q_c is to simply initialize S_c with $S_{R,c}$. Clearly, however, that idea has a major drawback that is: the results in $S_{R,c}$ might exhibit high degree of redundancy. This will lead to a selection of many non diverse results in S_c. To further explain that point, notice that the results in each $S_i \in S_{O,c}$ are diverse in their own. However, this assumption breaks once combining some of those diverse results together into $S_{R,c}$. This is because the results in two different sets S_i and S_j might be very similar to each other if their corresponding queries explored roughly the same data subspace.

For instance, Figure 2 shows result sets of three overlapping queries. As shown in figure, X_c is the result of a newly submitted query Q_c, which overlaps with two other historical results X_1 and X_2. The diversified set S_1 extracted from X_1 is shown in squares, whereas the diversified set S_2 extracted from X_2 is shown in triangles. The diverse results from both queries that overlap with X_c are retrieved and form the set of reusable diverse results $S_{R,c}$. It is clear from Figure 2, however, that set $S_{R,c}$ may contain few results that are very similar to each other. Thus, initializing diverse subset S_c with $S_{R,c}$ will adversely affect the final diversity of set S_c.

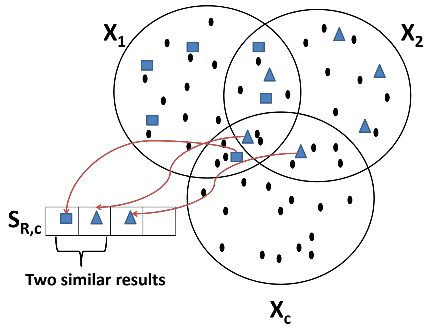

Figure 2: Reusable Set $S_{R,c}$ for current query result set X_c generated from diverse overlapping results of previous queries

Taking advantage of natural overlap occurring between various queries within and across user sessions and at the same time ensuring that only dissimilar results are selected to be included in the diverse subset of current query is precisely the goal of our DivIDE scheme. Next we present in detail how the DivIDE scheme achieves this goal.

3.3.1 Dividing the Search Space

Recall that in each iteration of the greedy construction algorithm, the one result that maximizes the diversity function is added to the diverse subset. In other words, in each iteration i, the result x_{max} with maximum set distance from the partially computed diverse subset S_c^{i-1} is selected for addition. Clearly, locating this optimal result requires computing the distance between each result in X_c from S_c^{i-1}. For large data sets where numerous results are returned in response to each query, and where near real time performance is expected, the cost of performing that search becomes computationally prohibitive. Hence, the goal of DivIDE is to minimize the cost incurred in locating the set S_c in the potentially large search space defined by the results X_c. To minimize that cost, DivIDE actually "*divides*" the search space (i.e., X_c) into two disjoint subsets, as follows:

1. **Reusable Diverse Results ($S_{R,c}$):** The set of diversified results of $\mathbb{Q}_{O,c}$ that overlap with the results in X_c. That is, as defined above, $S_{R,c}= X_c \cap S_{O,c}$.
2. **New Query Results (X_c'):** This is the set of results in X_c after excluding $S_{R,c}$. That is, $X_c' = X_c \setminus S_{R,c}$

As mentioned above, the set of results in $S_{R,c}$ have high potential to appear in the diversified set S_c. However, instead of blindly initializing S_c with $S_{R,c}$, DivIDE alternates between the two sets of results (i.e., X_c' and $S_{R,c}$) in an efficient manner and during that process it "selectively" chooses only the most promising results to be added in S_c so that to avoid compromising the quality of diversification. Notice that in each iteration i of the Greedy heuristic, we have a partially computed diverse subset S_c^{i-1} of size k_{i-1} such that $k_{i-1} \leq k$. The choice of next optimal result x_{max} is obviously made after examining all the candidate results in X_c.

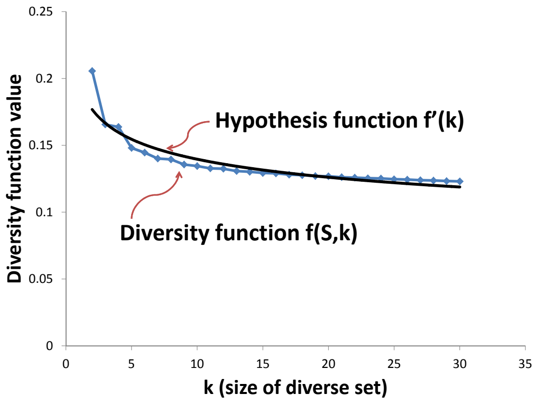

Figure 3: Diversity Curve.

In this work we take an alternate approach for identifying x_{max}. Instead of examining all the results in X_c, DivIDE aims to predict the maximum value of the diversity function $f(S^i, d)$ that can be achieved by including another result to S_c^{i-1} in advance. This value can then be leveraged to *prune* the search space. Hence, if $S_{R,c}$ contains the result that provides diversity value comparable to the estimated diversity value, then there is no need to search the query result set X_c', which reduces the search cost.

Clearly, several statistical and probabilistic models are applicable for estimation of diversity function. Such models have also been widely used for approximate query processing (e.g.,[9, 32]). This includes models for Gaussian processes, interpolation, regression, dynamic-probabilistic models, etc. In this work, we also adopt a *regression* model that is specifically suited to the diversification problem to efficiently and accurately estimate $f(S, d)$. Next we present the details of the regression model used by DivIDE for predicting diversity values.

3.3.2 Estimating Diversity

Regression models provide a powerful tool for investigation of relationships between variables. The basic idea of regression models is to relate a dependent variable V_D to an independent variable V_I. Nonlinear regression in particular is a good choice when there is an evidence to believe that the relationship between the dependent variable and independent variable follows a particular functional form ([25]). A nonlinear regression model has the form:

$$V_D = f'(V_I, \beta) + \epsilon$$

where V_D is dependent variable, f' is a known function of the independent variable V_I including parameters defined by β and ϵ is a random error ([25]).

In the case of diversification, the diversity value of a diverse set of results S clearly depends upon the number of results in that set (i.e., k). Specifically, small diverse subsets tend to exhibit higher diversity, whereas the value of diversity decreases with increasing the size of S. Hence, to ascertain the causal effect of diverse subset size k upon the value of the diversity function $f(S, d)$, we assume the value of the diversity function to be the dependent variable, where the diverse subset size is the independent variable.

To formally model the dependency between $f(S,d)$ and k for a certain database D, we use a *diversity curve* as shown in Figure 3. To generate that curve, a global sample query Q_G is executed over the database D to retrieve sample result set X_G (for more details on the different databases we have experimented with, please see Section 4). The Greedy heuristic is then applied over X_G to select a diverse subset S_G of size k_p. In each iteration i, where $i \leq p$, the diversity value $f(S_G^i, d)$ is evaluated and the pair $(k_i,\ f(S_G^i, d))$ is plotted. As Greedy finishes execution, p observations are plotted, as shown in Figure 3.

As Figure 3 shows, the diversity curve clearly exhibits a diminishing marginal gain trend. This is due to the fact that as new results are added using Greedy, the set of similar results already selected increases. Thus with each new addition to the diverse subset the marginal gain in diversity decreases, which is consistent with the study done in previous work [4]. Such trend allows us to use a simple model like power series to model the diversity curve (Figure 3).

Hence we can alternatively represent the diversity function $f(S,d)$ in terms of a computationally inexpensive hypothesis function $f'(k)$ such that:

$$Hypothesis\ Function\ f'(k) = ak^{-b}$$

where a and b are the parameters we seek that would best fit the function to the sample data.

Standard statistical calculations are used to determine the values of parameters a and b. Particularly, we use Least Squares estimation to measure the fitness of the regression model and Root Mean Square Error to measure the accuracy of predicted values. Other estimations can also be considered to accommodate the impact of outliers in data. The details of how parameter a and b are evaluated are given in appendix A.

Notice that the values of our model parameters a and b are learned from observation data. As mentioned above, this observation data is basically a set of pairs, where each pair is in the form $(k_i,\ f(S_G^i, d))$, evaluated over the results of a sample query Q_G. Hence, the range of these diversity values depends upon: (1) the size of the data subspace accessed by Q_G, and (2) the distribution of data within that subspace.

For instance, as shown in Figure 1, the data space covered by Q_1 is larger than the space covered by Q_4 and thus the diversity values evaluated for both queries, for same values of k, is expected to be different. Hence, if the regression model is built using data retrieved by Q_1 , the same model can only be used for Q_4 after scaling the regression coefficients a and b. Keeping this fact in mind, we build a global regression model over the data retrieved using a global query Q_G, which covers whole data space. This model is then scaled for every user query Q by a ratio of the data space covered by Q as compared to the space covered by Q_G. For very large data sets and where generating such global query is computationally very expensive, a good representative sample of the data can be used to build the global regression model.

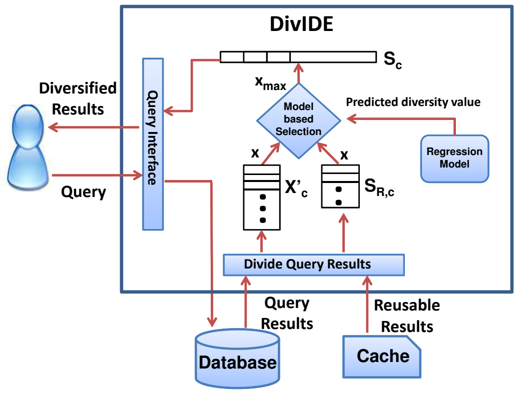

Figure 4: DivIDE Architecture.

3.3.3 *Prediction Using Regression Model*

As shown in Figure 4, to diversify a result set X_c, DivIDE divides X_c into two subsets: the reusable diverse result set $S_{R,c}$ and the new query result set X'_c. In each iteration i, an expected value of the diversity function is computed using the hypothesis function formulated above (i.e, $f'(i)$). To decide on the next result to be added to S_c^{i-1}, DivIDE uses a *deviation threshold* θ which is user-specified and a *deviation* value. The *deviation* value of any result $x \in X_c$ with respect to the diverse subset S_c^i is evaluated as:

$$deviation(x) = \left| f'(i) - f(S_c^{i-1} \cup \{x\}, d) \right|$$

Particularly, DivIDE selects a result $\bar{x}_{max}$, which has the deviation less than θ. For selecting that result $\bar{x}_{max}$, DivIDE uses the following two alternative selection methods:

- **Best Fit Diversification (DivIDE-BF):** Search the set $S_{R,c}$ for the result $\bar{x}_{max}$, which will maximize the diversity function value if added to set S_c^{i-1}. If the $deviation(\bar{x}_{max})$ is less than deviation threshold θ, then $\bar{x}_{max}$ is added to the set S_c^{i-1}. Otherwise, the optimal result x_{max} is located in X'_c and in turn, added to S_c^{i-1}.

- **First Fit Diversification (DivIDE-FF):** Search the set $S_{R,c}$ for the first result $\bar{x}_{max}$, which has the deviation value than deviation threshold θ. If such result is found, it is added to set S_c^{i-1}, else X'_c is searched to locate $\bar{x}_{max}$.

Clearly, each time a result is located in $S_{R,c}$, it saves at least $|X_c|$-$|S_{R,c}|$ number of distance calculations and data comparison operations. In particular, while searching $S_{R,c}$, DivIDE-BF evaluates the deviation value of the result with maximum set distance from the diverse subset S_c^{i-1}. In order to locate that result DivIDE-BF examines all the results in $S_{R,c}$. DivIDE-FF, however, picks the first result that provides comparable diversity to the one predicted by the regression model. Thus, DivIDE-FF further reduces the computational cost of diversification. The details of the DivIDE scheme is given in Algorithm 3.

Algorithm 3 DivIDE

Input: Query Result set $X_c = (x_1, x_2..., x_m)$, an integer k, Set of previous overlapping diverse results $S_{R,c}$, deviation threshold θ, a hypothesis function $f'(k)$

Output: Set S^k=(s_1, s_2...s_k) with the k most diverse items of X_c.

```
1:  $X'_c \leftarrow X_c \setminus S_{R,c}$
2:  $i \leftarrow 1$
3:  $x \leftarrow$a random result in $X_c$
4:  $S^i \leftarrow \{x\}$
5:  $i \leftarrow i + 1$
6:  while $i < k$ do
7:    if (Best Fit) then
8:      $\bar{x}_{max} \leftarrow argmax_{\bar{x}_{max} \in S_{R,c}} \sum_{x_j \in S^{i-1}} d(\bar{x}_{max}, x_j))$
9:      $S^i \leftarrow S^{i-1} \cup \{\bar{x}_{max}\}$
10:     if $(deviation(\bar{x}_{max}) > \theta)$ then
11:       $S^i \leftarrow S^{i-1} - \{\bar{x}_{max}\}$
12:       $x_{max} \leftarrow argmax_{x_{max} \in X'_c} \sum_{x_j \in S^{i-1}} d(x_{max}, x_j))$
13:       $S^i \leftarrow S^{i-1} \cup \{x_{max}\}$
14:     end if
15:   end if
16:   if (First Fit) then
17:     $\bar{x}_{max} \leftarrow x_j$ s.t $x_j \in S_{R,c}$, $deviation(x_j) \leq \theta$
18:     if $(\bar{x}_{max} = \phi)$ then
19:       $\bar{x}_{max} \leftarrow x_j$ s.t $x_j \in X'_c$, $deviation(x_j) \leq \theta$
20:     end if
21:     $S^i \leftarrow S^{i-1} \cup \{\bar{x}_{max}\}$
22:   end if
23:   $i \leftarrow i + 1$
24: end while
25: return $S^k$
```

Next we discuss managing the storage of historical diversified subsets (i.e., $\mathbb{S}_H$).

3.3.4 Cache Management

In an interactive exploration environment where multiple queries are generated within and across user sessions, it is very likely that after a while the size of the cache (i.e., $\mathbb{S}_H$) will grow to match the size of underlying data set. This will pose two challenges:

- As the size of cache increases, the search time for locating a reusable set $S_{R,c}$ for a query Q_c also increases.
- The size of the reusable set $S_{R,c}$ may grow to become larger than the size of X'_c.

As a consequence of the observations above, a large cache size would reduce the amounts of savings provided by DivIDE until it reaches zero. That is, the processing cost of DivIDE becomes similar to that of Greedy. The obvious approach to address these challenges is to limit the cache size so that to store only a limited number of diverse subsets. That is, instead of storing the entire history of diverse results $\mathbb{S}_H$, a subset $\mathbb{S}'_H$ of that history is stored Accordingly, as new queries are posed, some of the diverse subsets are evicted from cache to make room for new subsets. Let the number of cached diverse subsets at a particular time be denoted by L. Hence, when the number of diverse subsets become greater than L, DivIDE replaces one of the stored diverse subset.

Clearly, choosing the best cache replacement policy is application dependent and can be determined on the basis of query trends in a particular database application. In this work, we simply adopted the popular *Least Frequently Used (LFU)* cache replacement policy to evict the diverse subset that is accessed the least number of times. Studying the impact of other cache replacement policies is part of our future work.

Table 2: Evaluation Setting.

Parameter	**Range**	**Default**
Number of Queries (N)	2–1000	100
Diverse Subset Size (k)	10–100	30
% of Cached Queries (L)	10–50	20
Deviation Threshold (θ)	-	0.05
Data Size (D)	20k–40k	20k
Data sets	Unif., Clust., SDSS	Unif.

4. EXPERIMENTAL TESTBED

We perform a number of experiments to evaluate the efficiency and the effectiveness of our DivIDE scheme. In particular, we compare DivIDE against two baseline approaches: Greedy and SGC. Table 2 summarizes the different parameters used in our experimental evaluation.

Schemes: We evaluate the performance of following schemes:

- Greedy: Applies the Greedy Construction heuristic independently on each query result set to select the respective diverse subset (Section 3.1).
- SGC: Uses cached overlapping diverse results to initialize the diverse subset of a current query from its immediate predecessor query (Section 3.2).
- DivIDE: Uses a model-based approach to predict future values of the diversity function for the selection of reusable results from previous queries. We evaluate both best fit and first fit alternatives for this method (Section 3.3).

Performance Measures: The performance of each algorithm is measured based on the following metrics:

- Cost ($\sum_{i=1}^{N} C(S_i)$), measured as the sum of operations performed to evaluate diverse subsets of N queries, where each operation represents a distance computation and a comparison evaluation.
- Diversity ($\frac{1}{N} \sum_{i=1}^{N} D(S_i)$), measured as average diversity across the diversified subsets of N queries.

Data sets: We use both synthetic and real data sets. Our synthetic data sets consist of points in the 2-dimensional Euclidean space. Points are either uniformly distributed

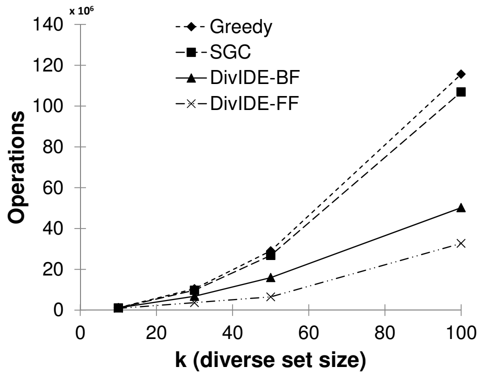

Figure 5: Impact of varying k on Cost

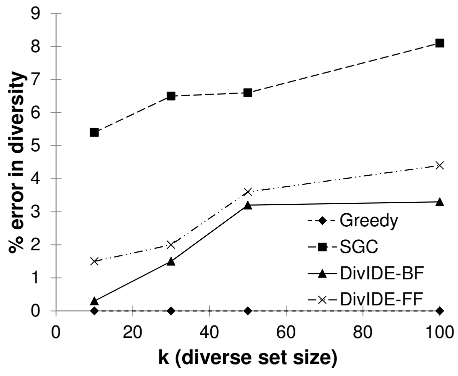

Figure 7: Maximum Diversity Error

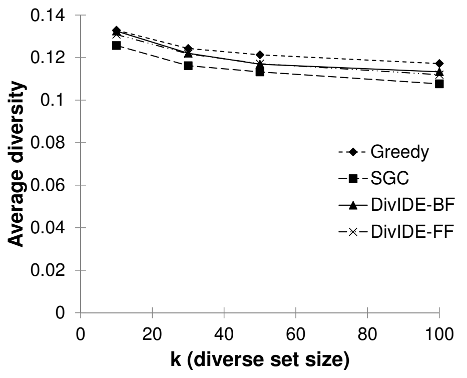

Figure 6: Impact of varying k on Diversity

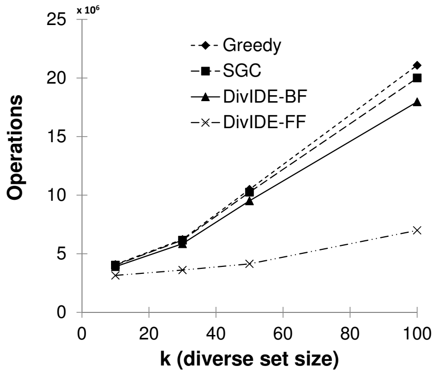

Figure 8: Impact of varying k on Cost (Clustered data set)

("Uniform") or form clusters around a random number of points ("Clustered"). Our real data set is based on the SDSS database and contains $40k$ data rows. We use the uniformly distributed numerical columns `rowc` and `colc` from PhotoObjAll table. For all data sets, attribute values are normalized to [0–1].

Queries: We simulate random user sessions with multiple range queries. For each experiment, the number of total queries N, across sessions is in the range [2–1000]. Each query is also associated with the size of diverse set k, which takes values in the range [10–100].

5. EXPERIMENTAL EVALUATION

In the following experimental results, we evaluate the sensitivity of DivIDE to the different parameters discussed in the previous section.

5.1 Impact of varying Diverse Set Size

In this experiment, we report on the impact of the required number of diverse results k. We first focus on one of our data sets, namely "Uniform". Figure 5 shows the number of operations (i.e., cost) performed by Greedy, SGC and the DivIDE schemes. As the figure shows, as the value of k increases, the cost increases for all schemes. DivIDE, however, is performing up to 70% less operations than Greedy. Meanwhile SGC reduces the cost by only 10% as compared to Greedy. This is due to the fact that SGC only utilizes a limited number of diverse results, which are obtained from the diverse subset of only one previous query.

Further, Figure 5 also shows that between the two variants of DivIDE, DivIDE-FF (first fit diversification) performs better in terms of cost as compared to DivIDE-BF (best fit diversification). In particular, DivIDE-FF reduces the cost by up to 15% compared to DivIDE-BF. This is clearly because DivIDE-FF terminates the search for an optimal result earlier without having to evaluate all the candidate results.

Figure 6 shows that the average diversity achieved by each scheme is decreasing with increasing the value of k. Both SGC and DivIDE achieve comparable average diversity to Greedy. However, DivIDE locates more diverse results than SGC as it uses regression model to avoid selecting very similar results. To highlight the benefits of DivIDE in terms of achieved diversity, in Figure 7 we plot the maximum loss in diversity incurred by each method as compared to Greedy. As the figure shows, the maximum loss in diversity for DivIDE is within only 4% compared to Greedy, whereas it goes up to 8% in the case of SGC.

Among the DivIDE methods, DivIDE-BF performs better in terms of achieved diversity (Figure 7). This is due to the fact that in each iteration, DivIDE-BF selects the one result providing the highest diversity if added to the diverse subset. Meanwhile, DivIDE-FF selects the first result that provides a diversity value within the threshold limit, thus introducing

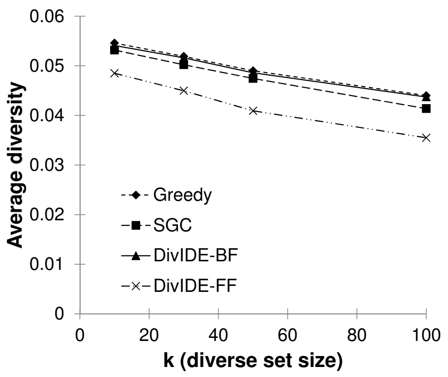

Figure 9: Impact of varying k on Diversity (Clustered data set)

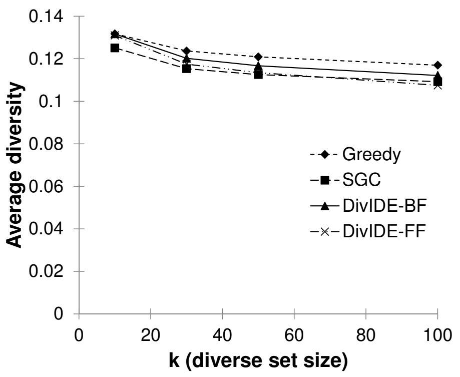

Figure 11: Impact of varying k on Diversity (SDSS data set)

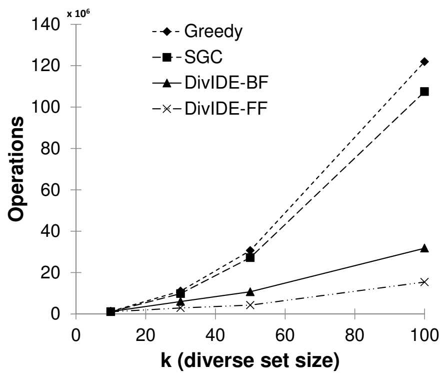

Figure 10: Impact of varying k on Cost (SDSS data set)

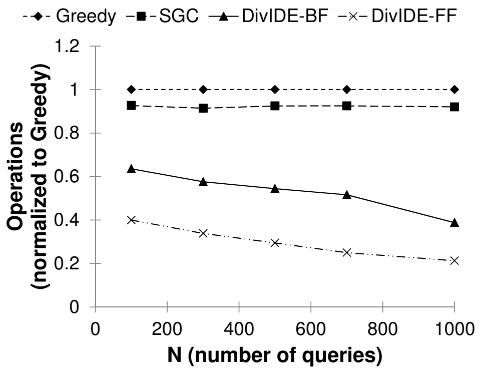

Figure 12: Impact of Number of Queries on Cost.

higher degree of approximation as compared to DivIDE-BF. However as the regression model is used to control the degree of approximation, the loss in diversity between the two methods is within 2%.

Next, we report on the performance of DivIDE scheme on the clustered and SDSS data sets.

Figure 8 and Figure 9 shows that DivIDE-BF method performs up to 15% less operations as compared to Greedy in achieving average diversity that is only 0.5% less than the diversity values achieved by Greedy. In comparison to the uniform data set, the savings in operations by DivIDE-BF method are quite less for clustered data. This is because it is expected that a regression model generated for a clustered data space would not perfectly fit all the different queries in that space, even after scaling. As a result, very few results in $S_{R,c}$ will fall within the deviation threshold θ. Hence, DivIDE-BF falls back to searching X'_c to locate the diverse results, which highly resembles Greedy. To the contrary, the performance of DivIDE-FF in terms of cost is mostly unaffected by the data distribution, but it has a slightly adverse effect on the quality of diversification.

SGC evaluates results with average diversity 6% less than the diversity of results selected by Greedy at the expense of only 5% savings in cost. SGC performs better in terms of quality of diversification for clustered data as less overlap among queries leads to less number of approximated results in the diverse subset of each query.

Figures 10 and Figure 11 show that for the SDSS real data set, the performance of DivIDE is comparable to its performance on the uniform data set.

It is clear from the experimental results that for different values of diverse subset size k, DivIDE outperforms Greedy and SGC algorithm in terms of number of operations with negligible loss in achieved diversity.

5.2 Impact of Number of Queries

In this experiment, we vary the number of queries, while keeping the diverse subset size at $k = 30$. Figure 12 shows that the total number of operations increase for all schemes with increasing the number of queries. However, the cost savings achieved by DivIDE also increase for higher values of N. This is because diverse results from previous queries are now shared among larger number of future queries. As N approaches 1000, the cost savings provided by DivIDE are up to 80%. The performance of SGC is not affected by increasing the number of queries as it only uses the diverse results from one predecessor query.

As shown in Figure 13, DivIDE achieves diversity within 2% of that achieved by Greedy. That high diversity indicates that in the presence of a larger number of cached diverse

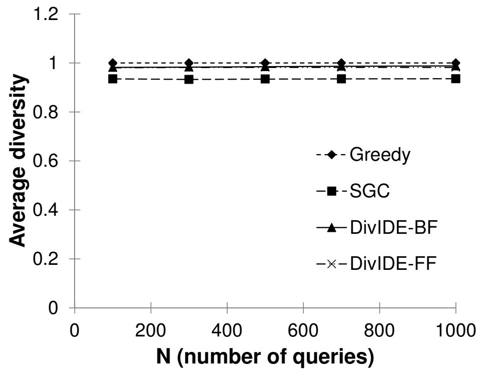

Figure 13: Impact of Number of Queries on Diversity.

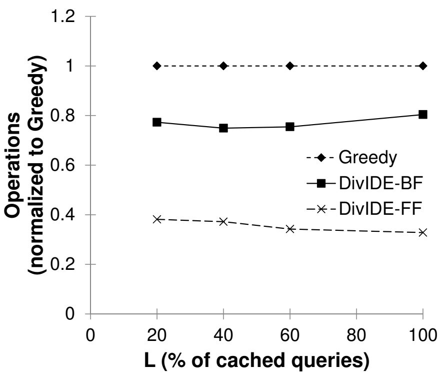

Figure 14: Impact of Cache Size on Cost.

results, the chances of finding approximated results that are close to the optimal results are higher.

5.3 Impact of Cache Size

To study the impact of cache size in terms of number of cached queries, we generate 500 random queries across various user sessions. Then, we vary the percentage of cached queries L from 10% of the total number of queries to 100%.

Notice that in this experiment we are only evaluating the performance of DivIDE against Greedy, as SGC is not affected by the specified cache size. In terms of cost (i.e., number of operations), DivIDE-BF exhibits an interesting pattern as shown in Figure 14. In that pattern, the cost of DivIDE-BF decreases as the cache size increases up to a point, after which it starts increasing. This is because at a moderate cache size, DivIDE-BF has a higher chance to find a cached result that is close to optimal, while at the same time incurring a relatively low overheard in searching the cache. As the cache size increases, DivIDE-BF still finds a cached result that is close to optimal, but it incurs a much higher cost in searching the rather large cache. DivID-FF shows a similar pattern ,but is more resilient to the cached results as it terminates the search early. Finally, Figure 15 shows that DivIDE consistently achieves diversity comparable to Greedy algorithm when varying the number of cached queries.

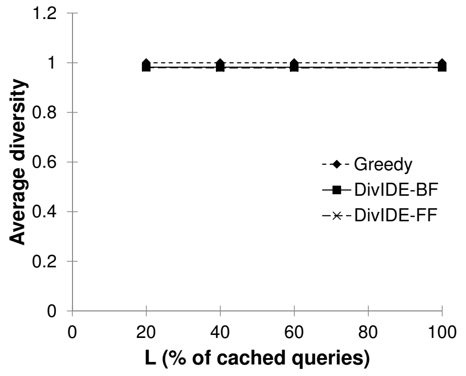

Figure 15: Impact of Cache Size on Diversity.

6. CONCLUSIONS

In this paper, we proposed the *(Diversification for Interactive Data Exploration) DivIDE* scheme. DivIDE targets the problem of efficiently diversifying the results of multiple queries within and across different exploratory sessions. Our novel scheme leverages the overlap between query results and utilizes a model-based approach that is particularly suitable for the efficient and effective diversification in IDE. DivIDE provides solutions of quality comparable to existing baseline solutions, while significantly reducing processing costs. We present experimental results concerning the efficiency and the effectiveness of our approach on both synthetic and real data sets.

7. REFERENCES

[1] R. Agrawal et al. Diversifying search results. In *WSDM*, 2009.

[2] S. Agrawal et al. Automated ranking of database query results. In *CIDR*, 2003.

[3] A. Albarrak, M. A. Sharaf, and X. Zhou. Saqr: An efficient scheme for similarity-aware query refinement. In *DASFAA*, 2014.

[4] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. *CoRR*, abs/1203.6397, 2012.

[5] U. Çetintemel et al. Query steering for interactive data exploration. In *CIDR*, 2013.

[6] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *SSDBM*. 2009.

[7] G. Chatzopoulou et al. The querie system for personalized query recommendations. *IEEE Data Eng. Bull.*, 34(2), 2011.

[8] C. L. A. Clarke et al. Novelty and diversity in information retrieval evaluation. In *SIGIR*, 2008.

[9] A. Deshpande and S. Madden. Mauvedb: supporting model-based user views in database systems. In *SIGMOD*, 2006.

[10] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *SIGMOD*, 2014.

[11] M. Drosou and E. Pitoura. Diversity over continuous data. *IEEE Data Eng. Bull.*, 32(4), 2009.

[12] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 39(1), 2010.

[13] M. Drosou and E. Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *PVLDB*, 6(1):13–24, 2012.

[14] M. Drosou and E. Pitoura. Dynamic diversification of continuous data. In *EDBT*, 2012.

[15] E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1), 1990.

[16] E. Erkut, Y. Ülküsal, and O. Yeniçerioglu. A comparison of *p*-dispersion heuristics. *Computers & OR*, 21(10), 1994.

[17] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[18] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

[19] M. L. Kersten et al. The researcher's guide to the data deluge: Querying a scientific database in just a few seconds. *PVLDB*, 4(12), 2011.

[20] H. A. Khan, M. Drosou, and M. A. Sharaf. Dos: an efficient scheme for the diversification of multiple search results. In *SSDBM*, 2013.

[21] H. A. Khan, M. Drosou, and M. A. Sharaf. Scalable diversification of multiple search results. In *CIKM*, 2013.

[22] E. Minack, W. Siberski, and W. Nejdl. Incremental diversification for very large sets: a streaming-based approach. In *SIGIR*, 2011.

[23] C. Mishra and N. Koudas. Interactive query refinement. In *EDBT*, 2009.

[24] D. Nanongkai et al. Interactive regret minimization. In *SIGMOD Conference*, 2012.

[25] O'Reilly, S. Boslaugh, and P. Andrew. *Statistics in a Nutshell, a desktop quick reference (2. ed.)*. 2012.

[26] Parameswaran et al. Seedb: Visualizing database queries efficiently. *VLDB*, 7, 2013.

[27] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Facility dispersion problems: Heuristics and special cases. In *WADS*, 1991.

[28] P. Roy et al. Efficient and extensible algorithms for multi query optimization. In *SIGMOD*, 2000.

[29] A. D. Sarma et al. Beyond skylines and top-k queries: representative databases and e-commerce product search. In *CIKM*, 2013.

[30] S.Borzsony, D.Kossmann, and K.Stocker. The skyline operator. In *ICDE*, 2001.

[31] T. Sellam and M. L. Kersten. Meet charles, big data query advisor. In *CIDR*, 2013.

[32] M. A. Sharaf et al. Balancing energy efficiency and quality of aggregate data in sensor networks. *VLDB J.*, 13(4), 2004.

[33] Y. Tao, X. Xiao, and J. Pei. Efficient skyline and top-k retrieval in subspaces. *IEEE Trans. Knowl. Data Eng.*, 19(8), 2007.

[34] Q. T. Tran et al. Query by output. In *SIGMOD*, 2009.

[35] M. Vartak, V. Raghavan, and E. A. Rundensteiner. Qrelx: Generating meaningful queries that provide cardinality assurance. In *SIGMOD*, 2010.

[36] M. R. Vieira et al. On query result diversification. In *ICDE*, 2011.

A. APPENDIX: REGRESSION MODEL

In statistics, nonlinear regression is a form of regression analysis in which observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables. The data are fitted by a method of successive approximations. Examples of nonlinear functions include exponential functions, logarithmic functions, trigonometric functions, power functions, Gaussian function, and Lorenz curves.

In this paper we have used power function that can be represented as: $y = ax^{-b}$. a and b are the parameters we seek that would best fit the function to the sample data. These two parameters can be determined by using *Non-linear least squares* analysis that is used to fit a set of m observations with a model that is non-linear in n unknown parameters (m > n). The basis of the method is to approximate the model by a linear one and to refine the parameters by successive iterations

Thus the equation for squared error for N sample observations is presented as:

$$E^2 = \sum_{i=1}^{N} (y_i - f'(x_i))^2$$

The two parameters are found by partial derivative equations:

$$\frac{\partial E^2}{\partial a} = 0$$

$$\frac{\partial E^2}{\partial b} = 0$$

which are solved simultaneously to obtain:

$$b = \frac{\sum x_i ln(y_i) - \frac{1}{N}(\sum x_i)(\sum ln(y_i))}{(\sum x_i^2) - \frac{1}{N}(\sum x_i)^2}$$

$$a = exp\left[\frac{1}{N}\sum ln(y_i) - b\frac{\sum x_i}{N}\right]$$

It has been shown that this yields a Coefficient of Determination of:

$$r^2 = \frac{\left[\sum x_i ln(y_i) - \frac{1}{k}(\sum x_i)(\sum ln(y_i))\right]^2}{\left[\sum x_i^2 - \frac{(\sum x_i)^2}{k}\right]\left[\sum ln(y_i)^2) - \frac{(\sum ln(y_i))^2}{k}\right]}$$

As in linear regression case, a value of $r^2 = 1$ infers a '"good fit"' of the model to the data.