THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Scalable Diversification for Data Exploration Platforms

Hina Anwar Khan

Masters of Computer Science

A thesis submitted for the degree of Doctor of Philosophy at

The University of Queensland in 2016

School of Information Technology & Electrical Engineering

Abstract

Today, data exploration platforms are widely used to assist users in locating interesting objects within large volumes of scientific and business data. In those platforms, users try to make sense of the underlying data space by iteratively posing numerous queries over large databases. Hence, data exploration platforms rely on methods for the extraction of *representative data* to provide users with a concise and meaningful representation of query results. That is, extracting a few tuples from a query result to provide quick insights in the potentially huge answer space. In the past few years, importance of *diversification* while extracting representative subsets of data has been greatly emphasized. It has been shown that diverse subsets provide more effective representation of the underlying data by minimizing redundancy and increasing coverage. Meanwhile, search results diversification adds additional cost to an already computationally expensive exploration process.

In this PhD thesis, we have focused on the design, implementation and evaluation of *scalable diversification* algorithms and schemes for the data exploration platforms. Particularly, this research stipulates that extracting diverse representative results during data exploration requires addressing several challenges including: 1) scaling to big volumes of high dimensional data, 2) large number of users, and 3) enabling real time continuous exploration. To address those challenges we focus on two broad aspects: 1) Diversification of high dimensional large data sets and 2) Diversification of multiple user queries.

The existing work conducts diversification in two steps: first compute all relevant query results, and then diversify the query results to select a small diverse subset. Similarly, all the dimensional attributes of all the data points in a query result are considered for diversification. Such a generic approach would be a performance bottleneck in high-dimensional large databases. To efficiently compute diverse subsets of query results exhibiting both high dimensionality and high-cardinality, we have proposed the *Progressive Diversification Scheme*. Our proposed scheme, utilizes the partial distance computations to reduce the amount of CPU and I/O incurred during query diversification. Moreover, to avoid the overhead of computing all relevant results first, we propose embedding diversification in query evaluation step by utilizing column-based data storage systems. In addition to computational cost of diversification methods, we have also considered the complexity of diversity objective function in high dimensional databases. Often, computing diverse solutions along all the dimensions is not a realistic approach. In fact, users may have some pre-specified preferences over some dimensions of the data, while expecting good coverage over the other dimensions. Motivated by that need, we propose a novel scheme, which aims to generate representative data that balance the tradeoff

between regret minimization and diversity maximization. Our scheme is based on a hybrid objective function that combines both regret and diversity. Additionally, it employs several algorithms that are designed to maximize that objective function.

We also address the diversification of multiple queries across and within user sessions. While, most of the current work is focused on the diversification of a single query result, we have proposed two scalable schemes for the diversification of both concurrent and sequential multiple queries. The results of the concurrent queries are available simultaneously and hence, provide an opportunity to exploit the overlap in those results. Our proposed algorithms leverage the natural overlap in search results in conjunction with the concurrent diversification of those overlapping results. In order to further reduce the processing costs of diversification, we have employed various approximation techniques that provide orders of magnitude reductions in cost, while maintaining a quality of diversification comparable to that of near optimal schemes. In contrast to the concurrent queries across user sessions, the queries within a single session are submitted sequentially at different intervals of time. Therefore, we have proposed a sequential diversification scheme that exploits the properties of data diversification functions while leveraging the natural overlap occurring between the results of different queries. Our proposed scheme relies on a regression model-based diversification method and an order based cache. In particular, we employ an adaptive regression model to estimate the diversity of a diverse subset. Such estimation of diversity value allows us to select diverse results without scanning all the query results. In order to further expedite the diversification process, we propose an order-based caching scheme to leverage the overlap between sequence of data exploration queries.

Our extensive experimental evaluation on both synthetic and real data sets shows the significant benefits provided by our proposed schemes as compared to existing diversification methods.

Declaration by Author

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly-authored works that I have included in my thesis.

I have clearly stated the contribution of others to my thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, and any other original research work used or reported in my thesis. The content of my thesis is the result of work I have carried out since the commencement of my research higher degree candidature and does not include a substantial part of work that has been submitted to qualify for the award of any other degree or diploma in any university or other tertiary institution. I have clearly stated which parts of my thesis, if any, have been submitted to qualify for another award.

I acknowledge that an electronic copy of my thesis must be lodged with the University Library and, subject to the policy and procedures of The University of Queensland, the thesis be made available for research and study in accordance with the *Copyright Act 1968* unless a period of embargo has been approved by the Dean of the Graduate School.

I acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material. Where appropriate I have obtained copyright permission from the copyright holder to reproduce material in this thesis.

Publications during candidature

- Hina A. Khan, Mohamed A. Sharaf: *Progressive Diversification for Column-based Data Exploration Platforms.* In the Proceedings of 31^{st} IEEE International Conference on Database Engineering (ICDE), APril 2015.
- Zaeem Hussein, Hina A. Khan, Mohamed A. Sharaf: *Diversity with few regrets: yet too few to mention.* In the Proceedings of 2^{nd} International workshop on Exploratory search in Databases and Web (ExploreDB), May 2015.
- Hina A. Khan, Mohamed A. Sharaf, Abdullah Albarrak: *DivIDE: efficient diversification for Interactive data exploration.* In the proceedings of 26^{th} International Conference on Scientific and Statistical Database Management (SSDBM), July 2014.
- Abdullah Albarrak, Tatiana Noboa, Hina A. Khan, Mohamed A. Sharaf, Xiaofang Zhou, Shazia Wasim Sadiq: *ORange: Objective-Aware Range Query Refinement.* In the proceedings of 15^{th} IEEE International Conference on Mobile Data Management (MDM), July 2014.
- Hina A. Khan, Marina Drosou, Mohamed A. Sharaf: *Scalable diversification of multiple search results.* In the proceedings of 22^{nd} ACM Conference on Information and Knowledge Management (CIKM), October 2013.
- Hina A. Khan, Marina Drosou, Mohamed A. Sharaf: *DoS: an efficient scheme for the diversification of multiple search results.* In the proceedings of 25^{th} International Conference on Scientific and Statistical Database Management (SSDBM), july 2013.

Publications included in this thesis

Hina A. Khan, Mohamed A. Sharaf: *Progressive Diversification for Column-based Data Exploration Platforms.* In the Proceedings of 31^{st} IEEE International Conference on Database Engineering (ICDE), APril 2015. (Chapter 3)

Contributor	Statement of contribution
Hina A. Khan (candidate)	Designed algorithms and experiments (100%), Paper Writing (60%)
Mohamed A. Sharaf	Discussion and analysis of the algorithm design, Paper Writing (40%)

Hina A. Khan, Marina Drosou, Mohamed A. Sharaf: *Scalable diversification of multiple search results.* In the proceedings of 22^{nd} ACM Conference on Information and Knowledge Management (CIKM), October 2013. (Chapter 4)

Contributor	Statement of contribution
Hina A. Khan (candidate)	Designed algorithms and experiments (100%), Paper Writing (50%)
Marina Drosou	Discussion and analysis of the algorithm design, Paper Writing (20%)
Mohamed A. Sharaf	Discussion and analysis of the algorithm design, Paper Writing (30%)

Hina A. Khan, Marina Drosou, Mohamed A. Sharaf: *DoS: an efficient scheme for the diversification of multiple search results.* In the proceedings of 25^{th} International Conference on Scientific and Statistical Database Management (SSDBM), july 2013. (Chapter 4)

Contributor	Statement of contribution
Hina A. Khan (candidate)	Designed algorithms and experiments (100%), Paper Writing (60%)
Marina Drosou	Discussion and analysis of the algorithm design. Paper Writing (10%)
Mohamed A. Sharaf	Discussion and analysis of the algorithm design, Paper Writing (30%)

Hina A. Khan, Mohamed A. Sharaf, Abdullah Albarrak: *DivIDE: efficient diversification for Interactive data exploration.* In the proceedings of 26^{th} International Conference on Scientific and Statistical Database Management (SSDBM), July 2014. (Chapter 5)

Contributor	Statement of contribution
Hina A. Khan (candidate)	Designed algorithms and experiments (100%), Paper Writing (60%)
Mohamed A. Sharaf	Discussion and analysis of the algorithm design, Paper Writing (30%)
Abdullah Albarrak	Discussion and feedback, Paper Writing (10%)

Zaeem Hussein, Hina A. Khan, Mohamed A. Sharaf: *Diversity with few regrets: yet too few to mention.* In the Proceedings of 2^{nd} International workshop on Exploratory search in Databases and Web (ExploreDB), May 2015. (Chapter 6)

Contributor	Statement of contribution
Zaeem Hussein	Algorithms design (50%) Experiments (100%), Paper writing (30%)
Hina A. Khan (candidate)	Algorithms design (50%), Paper writing (30%)
Mohamed A. Sharaf	Discussion and analysis of the algorithm design, Paper writing (40%)

Contributions by others to the thesis

My principle advisor, Dr. Mohamed Sharaf, has contributed towards the research presented in this thesis by providing guidance for problem formulation and solution refinement. He has also reviewed and polished the published papers included as part of this thesis.

Statement of parts of the thesis submitted to qualify for the award of another degree

None.

Acknowledgements

"The greatest of empires, is the empire over one's self" said Publilius Syrus and at no point in my life have I understood this sentence better than while completing my PhD. Pushed to my limits and then finding room still for greater improvement. This PhD process has changed me as a person. However, I could never have done it without the supportive network around me that enabled me to always get up when I felt down, believed in me when I did not and enabled me to give my best. My loving parents, caring husband and beautiful daughters are the chains of this network that held me together these past four years. I thank them from the bottom of my heart.

I would also like to acknowledge the beacon of support my advisor, Dr. Mohamed Sharaf has provided from day one. His patience, inspirational guidance, knowledge and constant encouragement have been instrumental in getting this thesis to completion. Many a times it was only his kindness, willingness to listen and wisdom that kept me going. I feel very lucky to have found a great mentor in him. It was an honour to work with him.

I am also very grateful to my associate advisor, Prof. Shazia Sadiq for her valuable insights and guidance. Her constant appreciation and encouragement always helped in building my confidence. I would also like to express my gratitude to my committee members, Prof. Xiaofang Zhou, Dr. Marcus Galagher and Mr. Hoyoung Jeung for their extremely useful feedback and evaluation of my research work throughout my PhD.

I want to thank the co-authors of my papers: Marina Drosou, Abdullah Albarrak and Zaeem Hussein for their hard work and knowledge sharing. I also thank all my colleagues and friends at Data and Knowledge Engineering (DKE) group for their support. It has been a memorable journey and I will always cherish these last four years.

Keywords

Data Exploration, Results Diversification, Query Optimization, Representative Data

Australian and New Zealand Standard Research Classifications (ANZSRC)

ANZSRC code: 080604, Database Management, 100%

Fields of Research (FoR) Classification

FoR code: 0806, Information Systems, 100%

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Data exploration is a key ingredient in a widely diverse set of discovery-oriented applications including the ones from science, business and finance [15,54,78]. In those applications, generally users explore their data interactively for locating interesting objects. This data discovery process is typically ad-hoc and labor intensive as: i) datasets are complex and heterogeneous, ii) the users come from diverse backgrounds with different understanding of the underlying datasets, and iii) the users may not have a prior notion of interesting objects. 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 [54]. Both professional and amateur astronomers access the SDSS archive. The professional users typically pose fairly complex queries on position, colors and other attributes of sky objects. Whereas, the amateur users initially explore the datasets with simpler queries and as they learn more about the detailed properties of the stars and galaxies, they are expected to define more sophisticated queries. The most common queries against SDSS database are spatial queries involving a small region in the sky [8]. For instance, consider the following example:

Example 1. *Assume an amateur user, who is curious about finding a region in the sky having group of objects with some interesting features. To locate those objects the user needs to access the PhotoObj table in SDSS that has around 500 attributes, most of which are floating point numbers, and more than 600 million rows. Those attributes define various features of stars, galaxies and sky samples stored in SDSS. However, the user has no prior information about which of those attributes and their corresponding values will make an object interesting. Hence,*

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

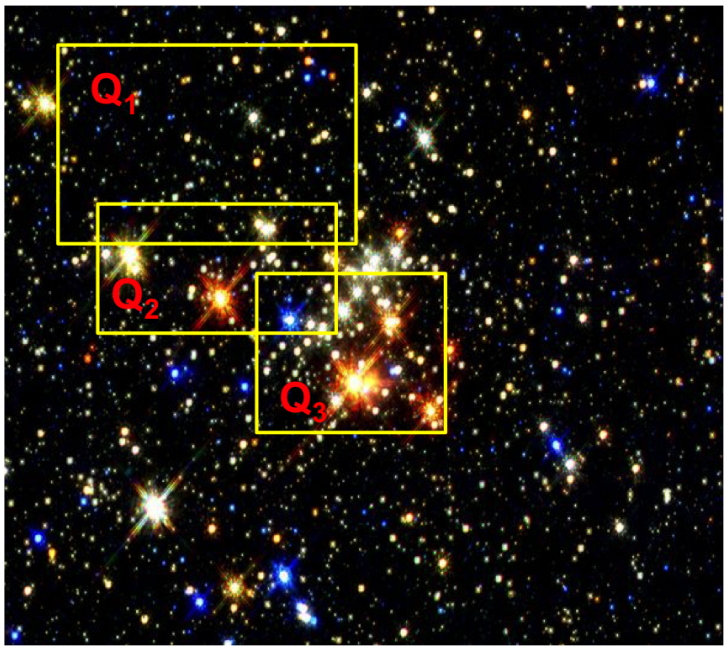


FIGURE 1.1: Series of exploratory queries generated in a user session

the user will start querying the database with some intuitive coordinate values for specifying a certain area of the sky. As shown in Figure 1.1, the user will first pose query Q_1 and review all the results returned by Q_1. Guided by the results returned by Q_1, the user may refine the query predicates to pose another query Q_2. The result set of each query enhances the user's understanding of the underlying data and allow him to pose a more precise query. This iterative process continues until the user finally finds a region of interest, which is represented by query Q_3 in Figure 1.1.

It should be clear from the example 1 that before locating the interesting results, a user may execute multiple exploratory queries that are initially vague and imprecise. Those queries potentially return thousands of results. It is essential for a user to review those results to assess their interestingness and refine their query based on them. However, comprehending large number of results is a time consuming and challenging task. Therefore, to reduce the user effort and overall exploration time, Data exploration platforms rely on several *representative data extraction techniques* to provide a synoptic understanding of the underlying query result. In particular, few representative results help user have a quick understanding of what is available in the whole result set and significantly reduce their result review time .

Thus, in the past years researchers have focused on developing effective representative data extraction methods. Beyond the well-studied top-k (e.g., [37, 51]), skyline (e.g., [13, 92]), sampling (e.g., [3–5]) and clustering techniques (e.g., [9,61,84]), Diversification is rapidly becoming the technique of choice for extracting representative subsets with high coverage and minimum

redundancy.

During a Data exploration session, a user query is first processed against the stored database, and the corresponding results are generated. The results of user queries are in the form of tuples that can be generally viewed as a set of attribute values represented as data points in a multi-dimensional data space. Those results are then further processed to compute a small diverse subset. 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 [97]. 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., [97]), (ii) novelty based, i.e., selecting results that contain new information when compared to what was previously presented to the user (e.g., [21]) and, (iii) semantic coverage based, i.e., selecting results that belong to different categories or topics (e.g., [6]).

In this work, we primarily focus on the widely used content-based definition of diversity, whose objective is to maximize the overall dissimilarity within a set of selected results. In particular, given a distance measure between two results, e.g., the Euclidean distance, the diversity of a set S is measured by a *diversity function* that captures the dissimilarity between the results in S based on either the *average* or the *minimum* of the pairwise distances between results [31, 97].

Since, computing a diverse subset of query result is a combinatorial optimization problem, identifying an optimal diverse subset S^* has been shown to be NP-hard ([35]). Hence, due to the high computational cost of identifying an optimal diverse subset, approximation methods are typically employed to select a near optimal diverse subset. However, the existing approximation algorithms have a computational complexity linear to the input data size. For small datasets, those algorithms are efficient but they fall short in scaling to data-intensive and computationally expensive data exploration tasks. Thus, achieving effective and efficient diversification in data exploration platforms still remains a challenging task. Therefore, this PhD thesis is aimed at developing scalable and efficient diversification schemes that are particularly suited to Data Exploration platforms. Next, we present the contributions of this thesis towards that goal.

1.2 Thesis Contributions

While diversification provides users with quick insights into the query answers, it adds additional complexity and requires extra computational resources. Thus, motivated by the need

to efficiently provide users with effective insights during data exploration, we have addressed the scalability of diversification algorithms at two levels: 1) Query Level: optimizing the diversification of a single query result in large high dimensional datasets, and 2) Session Level: optimizing the diversification of multiple query results in an exploratory session.

1.2.1 Query Level Optimizations

Exploratory queries are imprecise by nature and potentially return large result sets. Meanwhile, current techniques to data diversification follow a general *process-first-diversify-next* approach, in which a query is first executed on the database to generate a result which is then diversified to compute a small diverse subset. One drawback of that approach is that all the data points (i.e., tuples) in a query result are accessed from disk while only few diverse results are needed by the user. Similarly, all the dimensional attributes of all the data points in a query result are considered for diversification. However, only very few of those points will be included in the diverse set, while most of the remaining points will be discarded. Clearly, that generic approach would hinder the performance of applying diversification in high-dimensional large databases, in which results returned by queries exhibit both high-dimensionality and high-cardinality.

Apart from computational complexity of diversification algorithms in high dimensional spaces, the complexity of diversification objective function also need to be considered. Current diversification approaches present results diversified along all dimensions. In many applications, however, the user might require coverage along some dimensions while have some notion of preference associated with other dimensions of the data. In that case, it is desired to select representatives that: 1) maximize diversity over the dimensions targeted for coverage(i.e., low redundancy), and 2) minimize regret over the preference dimensions (i.e., high utility).

To address the aforementioned problems, we have proposed two novel schemes [48, 57]:

- **Progressive and integrated Diversification Scheme:** We propose the *Progressive Data Diversification (pDiverse)* scheme [57]. The main idea underlying pDiverse is to utilize *partial distance* computation to reduce the amount of CPU and I/O incurred during query diversification. In a traditional approach where diversification is decoupled from query processing, our scheme allows to quickly detect and prune those data points in the query result that cannot be included in the final diverse set. The early pruning of those points provides significant reduction in the CPU cost required for distance computations. Moreover, we propose integrating data diversification with query processing, which enables pushing down partial distance computation closer to the raw data, and

hence provides pruning at the data storage layer. This allows for saving the I/O costs incurred in accessing those pruned values, especially in vertically partitioned data (i.e., column-stores). In particular, *Column-store* systems vertically partition a database into a collection of individual columns, which are stored separately. Hence, instead of reading all the attribute values of all the data points accessed by a query, pDiverse leverages a column-based storage to selectively read only those attribute values of the unpruned data points and save I/O cost. We further optimize pDiverse by incorporating novel techniques for ordering of dimensions and approximation of diversity. Our extensive experimental evaluation on real and synthetic data sets illustrate the benefits achieved by pDiverse.

- **Diversity with Regret minimization:** Diversity maximization has been adopted as one technique to generate representative data with high coverage and low redundancy. Orthogonally, regret minimization has emerged as another technique to generate representative data with high utility that satisfy the user's preference. In reality, however, users typically have some pre-specified preferences over some dimensions of the data, while expecting good coverage over the other dimensions. Motivated by that need, we propose a novel scheme called ReDi, which aims to generate representative data that balance the tradeoff between regret minimization and diversity maximization [48]. ReDi is based on a hybrid objective function that combines both regret and diversity. Additionally, it employs several algorithms that are designed to maximize that objective function. We perform extensive experimental evaluation to measure the tradeoff between the effectiveness and efficiency provided by the different ReDi algorithms.

1.2.2 Session Level Optimizations

During Data exploration, the user interaction with the database takes the form of an exploratory session, or session for short. In a session, a user issues a sequence of related queries, in which each query serves as a springboard to the next. Hence, it is essential to present a diverse representative subset of query result to help user quickly formulate their next query. However, the computational cost of diversifying the results of multiple queries within an exploratory session, increases linearly with the increase in the session length (i.e., number of queries). The diversification process becomes even more computationally challenging as Data exploration platforms host multiple users running multiple sessions simultaneously. Meanwhile, delivering near real time performance remains an essential requirement for Data Exploration platforms so that to match the intrinsic nature of interactive and iterative data exploration to ensure user

satisfaction. Hence, there is a need for efficient diversification methods in Data Exploration platforms which are scalable to: i) Number of user sessions, and ii) Length of each user session. Towards that end, we propose novel schemes for diversification of multiple concurrent and sequential queries [55, 56, 58].

- **Diversification for Multiple Concurrent queries:** We propose the DivM (Diversification of Multiple Search Results) scheme that addresses the problem of efficiently diversifying the results of multiple concurrent queries across different user sessions. Towards this goal, DivM leverages the natural overlap in search results in conjunction with concurrent diversification of those results using partial aggregation techniques. This enables DivM to provide the same quality of diversification as that of the sequential methods, while significantly reducing the processing costs. We further generalize and extend the DivM scheme to exploit various approximation techniques that provide orders of magnitude reductions in processing cost, while maintaining a quality of diversification comparable to that of near optimal schemes. Our extensive experimental evaluation on both real and synthetic data sets shows the scalability exhibited by our proposed scheme under various workload settings, and the significant benefits it provides compared to existing methods.

- **Diversification for Multiple Sequential queries:** A user typically execute numerous related queries during a data exploration session. Unlike the concurrent queries executed across different sessions, the queries within a user session are executed in a sequence. Hence, the natural overlap in results of different queries in not known in advance. Therefore, to optimize the diversification of multiple sequential queries within a user session, we propose an efficient Diversification Scheme. Our proposed scheme, called *AdOr*, relies on two main interrelated components, namely: 1) an adaptive model-based diversification method, and 2) an order-based caching scheme. In particular, *AdOr* employs an adaptive model based diversification method to estimate the diversity of a diverse subset and hence selects diverse results without scanning all the query results. In order to further expedite the diversification process, *AdOr* employs an order-based caching scheme to leverage the overlap between sequence of data exploration queries. 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 *AdOr*.

1.3 Thesis Layout

The rest of the thesis is organized as follows: In Chapter 2, we present preliminaries and related work. In Chapter 3, we present the progressive diversification scheme for high dimensional large datasets and an integrated model that combines the processing of a range query with the diversification of its results in column databases. In Chapter 4, we formulate the multiple search results diversification problem and present novel algorithms to extract multiple diverse subsets concurrently. In Chapter 5, we present the interactive diversification scheme for session based exploratory search. In Chapter 6, we formulate a hybrid objective function to combine diversity and regret minimization, and present efficient algorithms to evaluate representative subsets based on that combined objective function. Finally, Chapter 7 concludes this thesis and overviews future work.

Chapter 2

Preliminaries and Related Work

2.1 Data Exploration

Traditional Database Management Systems are well suited for applications in which the user queries are precise and well understood. However, users of many discovery-oriented applications, including ones from scientific computing, financial analysis and evidence-based medicine, are searching for interesting insights in the data without a prior knowledge of what they are exactly looking for [15]. In short, user queries are imprecise and vague. Hence, *Exploratory Search* is ascending to a new level of importance because of its indispensible role in many discovery-oriented applications [54, 78]. In contrast to traditional database search, exploratory search process is inherently hard as users do not always have a clear idea of what they are looking for. Therefore, before locating interesting results, a user may execute large number of related queries.

For instance, Figure 2.1 shows a typical interaction of a user with a data exploration platform. Consider a database $\mathcal{DB}$, which consists of a set of D-dimensional tuples. Each tuple $x_i =< x_{i,1}, x_{i,2}, ..., x_{i,D} >$ is basically a point in a D-dimensional space, where the value of $x_{i,j}$

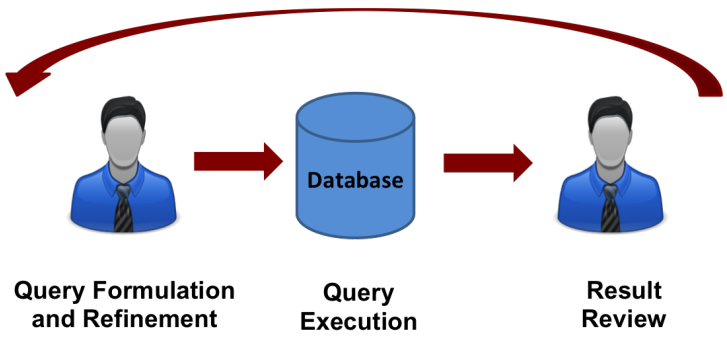


FIGURE 2.1: Data Exploration Session

is drawn from the domain of attribute A_j. User starts the exploration process by submitting a query. For instance let Q_0 be a range query submitted by a user. Q_0 is executed by the database system and the result set $X_0 = \{x_1, \ldots, x_n\}$, comprising the set of data points that fall within the multi-dimensional range specified by Q_0, is retrieved and presented to the user. If the user does not find the result set interesting, next query Q_1 is formulated by redefining the previous query Q_0. This marks the end of one *data exploration cycle* that can be defined as:

Definition 1. *Data Exploration Cycle: Sequence of steps performed to execute one user query. These steps involve query formulation/refinement, query execution and results review.*

Thus a user explores a D-dimensional database in multiple *data exploration cycles*. At the end of each cycle, the result set X_i to the user query Q_i is aimed to aid the user in understanding the data and provide guidance to formulate the next query Q_{i+1}. This exploration process continues till the user finds a result set that provides interesting insights. Let us call this process as one *data exploration session* that can be defined as follows:

Definition 2. *Data Exploration Session: A sequence of data exploration cycles performed by the user in order to extract one target result set.*

It should be clear from the discussion above that the data exploratory sessions can be long, laborious and computationally expensive. Hence, automated data exploration solutions have emerged to enable users to extract knowledge out of data with ease and efficiency (e.g., [15, 28, 50, 52, 69]). Here, we categorize the research efforts towards efficient data exploration along three phases of data exploration cycle namely: i) Query formulation/refinement, ii) Query execution, and iii) Result review.

- **Query Formulation/Refinement** : It is often difficult for users to pose precise queries without having good understanding of the data. Manually redefining queries after reviewing initial query results is a labor-intensive task. Thus, significant research work is aimed at assisting users formulate and refine their exploratory queries [2]. For instance, data driven systems are designed to define a query for users who are not aware of the exact query predicates but who are aware of data items relevant to their exploration task [70, 82, 94]. In those systems, queries are defined on the basis of the example output tuples. Recently, solutions for tuning imprecise queries, where the relevance of query predicates is uncertain to the user are also proposed [71]. In literature, many techniques are also proposed for query recommendation [38]. Beyond query formulation, in [28] a new

exploration interface is proposed that retrieves the relevant objects using only relevance feedback from users.

- **Query Execution** Once a query is formulated, the next key step in the data exploration cycle is the query execution. Since, an exploration session may comprise of many user queries it is important to reduce the execution time of each query in order to provide interactive performance. Data pre-fetching, caching and query approximation are important techniques used to reduce the query execution times. Recently, these techniques have been studied within the context of data exploration. For instance, data prefetching is used to optimize the spatial exploratory queries [93]. Cached results from previous queries are also used to generate the results of new queries [58]. Online query processing methods are used to generate approximate answers to user queries in order to give users a quick understanding of the actual query result set [45]. Another approach to generate approximate query results is to execute a query on a sampled data set. Many sampling techniques are proposed for generating quality query results within bounded execution times [4, 5, 83] .

- **Result Review**: The result set of each query acts as a dashboard for the next query in the data exploration session. Therefore it is very important that the query results are presented in a way that enhances user's understanding of the underlying data. As exploratory queries are often imprecise, query result set can be very large. Capturing interesting insights from large number of results is a challenging task even for expert users. Towards that end, data visualization and representative data extraction techniques have been shown to be effective in helping users comprehend the query results.

  Visual analytic tools such as Tableau [88], GGobi [90] and Improvise [98] help users to construct multi-dimensional views of data. Visualization systems that focus on displaying as many results as possible to provide users feedback as they refine their queries are also proposed [46, 53]. Besides displaying query results, some visualization systems are designed to recommend interesting data visualizations to users automatically [34, 69]

  Another approach to address the *too-many-results* problem is to present only representative query results. That is, extracting a few tuples from a query result to provide quick insights in the potentially huge answer space. That small representative subset of results should help users learn what is available in the whole result set and assist them in finding what they are looking for. One of the approaches to select the representative results is

to present *best* results with respect to some user preference criteria. Early examples of such representative data generation methods include the well-studied *top-k* and *skyline* queries [92]. In top-k, the user's preference is captured by means of a utility function over different dimensions of data, whereas in skyline, that preference is captured by applying the dominance property over those dimensions. Recently, regret minimization has been proposed as a practical alternative for both queries [66]. In regret minimization, a small representative set is generated by considering the universe of all possible utility functions.

However, often standard database query results comprise a set of tuples, with no associated ranking or preference [11]. Also, during data exploration the aim of representative result set is not to show the best results to users, rather help them understand what is available in the whole result set. Thus, in the absence of preference, clustering (e.g., [9, 61, 84]) and sampling (e.g., [3–5]) methods have been used to generate representative data. Cluster medoids can be viewed as representative results (e.g., [11]). However, clustering algorithms often ignore sparse data areas and tend to select more representatives from the dense areas. Similarly, data sampling techniques often lack the ability to discover outliers in data which for some applications may be indispensable. Therefore, *diversification* methods have been recently employed to generate representative sets that provide high coverage of the accessed data, while minimizing redundancy (e.g., [6, 31–33, 55, 56, 97]).

In this work, our focus is on scalable diversification of search results in data exploration platforms, which is motivated by the need to provide users with a diverse representative subset of results. A detailed overview of search results diversification is presented next.

2.2 Search Results Diversification

The goal of search result diversification, or diversification for short, is to return to the user the set of representative results that are not only relevant to the user query but are also *diversified*. The rationale behind such approach is to reduce the risk of *result redundancy* and maximize the coverage of query results. One of the earliest works emphasizing the importance of diversity in search results is presented in [14] as Maximal Marginal Relevance (MMR) problem. It shows how a trade-off between novelty and relevance of search results can be made explicit through the use of two different functions. The first measures the similarity among documents, and the other measures the similarity between documents and the query. Similarly in [101], it is stated

Symbol	Description
D	Dimensionality of the database
Q	User Query
X	Result set of Q
n	Size of result set X
S	Diverse subset of X
k	Diverse subset size
$f(S, d)$	Diversity function
$d(x_i, x_j)$	Distance between two points
$SetDist(x, S)$	Set distance between a point x and diverse subset S

Table 2.1: Table of Symbols.

that in general it is not sufficient to return a set of results relevant to the keyword query as the correlation among the returned results is also very important.

In the past years, diversification has been shown to be effective in many different domains; e.g., web search [6, 7, 97], recommender systems [86, 99, 100, 103], database search [63, 96] and query suggestions [24, 77, 85]. Regardless of the domain, in general the diversification problem can be defined as follows:

Definition 3. *Given a set X of n results and an integer k ($k \leq n$) that defines the size of the output set, select a subset S^* of X with $|S^*| = k$, such that the diversity of the results in S^* is maximized.*

There are various definitions of diversity in literature [97]. Most of them can be classified in one of the following categories:

- Content-based Diversity: Objective of Content-based diversity is to maximize the overall dissimilarity within a set of selected objects. (e.g., [97])
- Novelty-based Diversity: Novelty-based diversity emphasizes on selecting results that contain new information when compared to what was previously presented to the user (e.g., [21])
- Coverage-based Diversity: Coverage-based diversity focuses on selecting results that belong to different categories or topics (e.g., [6]). Hence, diversity is considered as a means for selecting results that cover many different interpretations of the informational need of a user.

(a) Sample Data (b) MaxSum (c) MaxMin

FIGURE 2.2: MaxSum vs MaxMin diverse solutions for n=40 and k=8

In this work, we primarily focus on the widely used content-based definition of diversity. Content-based diversity maximizes the overall dissimilarity within a set of selected objects [35]. 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 of size k 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 [97].

The most common diversity functions were originally studied in Operations Research for the *facility dispersion* problem [74]. More precisely, the problem concerned selecting locations for some dangerous or obnoxious facilities in order to make them mutually distant to each other. To address the *facility dispersion* problem, four different, but related, functions are proposed in [36]. The four functions are formulated in a general way, without reference to a specific solution space or distance metric. These formulations apply to the discrete version of the facility dispersion problem, where the locations for the new facilities are chosen from a finite set of candidate points in a multi-dimensional space, in order to maximize one of the following four functions:

- **MaxMinMin**: Maximize the minimum distance between each pair of facilities.
- **MaxSumMin**: Maximize the sum of the minimum distances from each facility to its closest neighbor.
- **MaxMinSum**: Maximize the minimum sum of the distances from each facility to all its neighbors.
- **MaxSumSum**: Maximize the sum of all the hub distances for all located facilities.

Among the four dispersion objective functions, the *MaxMinMin* and the *MaxSumSum* functions have been widely adopted for diversification of search results [42] and subject of much

research [60]. The *MaxMinMin* function is often referred to as the *MaxMin* diversity and the *MaxSumSum* function is more commonly known as *MaxSum* or *MaxAverage* diversity. The *MaxSum* diversity function tries to maximize the overall dissimilarity of the selected results, while *MaxMin* diversity function selects the results that are always separated by a minimum distance. Figure 2.2 shows a comparison of the results selected by both measures. The sample dataset is the data of swimming pools in Brisbane region comprising of ten attributes providing address and available facilities information of each pool (Figure 2.2(a)). The data is diversified only on the latitude and longitude attributes. As shown in Figure 2.2(b), the *MaxSum* function tends to focus more on the outskirts of the dataset. Whereas, *MaxMin* function picks results from the center as well and tends to provide more coverage (Figure 2.2(c)).

Formally, given a diverse subset S and a distance metric d, the *MaxMin* diversity function is defined as:

$$f(S,d) = \min_{\substack{x_i,x_j \in S \\ x_i \neq x_j}} d(x_i, x_j)$$

However, if the diversity is measured as the average distance among the results in S then the *MaxSum* diversity function is defined as:

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

Hence, the diversification of search results problem in general can be defined as follows:

Definition 4. *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, the Diversification problem is defined as selecting a subset S^* of X, such that:*

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

2.2.1 Diversification Algorithms

The Diversification problem has been shown to be NP-hard for both *MaxMin* and *MaxSum* diversity functions [25,36]. Since, a diverse set of size $k-1$ may not be a subset of diverse set of size k, it is not possible to design an incremental optimal solution. For instance, given a result set of size n, it is possible to find an optimal solution for $k = 2$ in $O(n^2)$ time. However, the solution for $k = 2$ cannot be used to build a solution of size $k = 3$. Hence, in order to solve the larger instances of diversification problem in polynomial time, many approximation algorithms

Algorithm 1 Greedy construction.

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: $t \leftarrow 1$
3: $S_t \leftarrow \{x\}$
4: $t \leftarrow t + 1$
5: **while** $t < k$ **do**
6: $\quad x_{max} \leftarrow \text{argmax}_{x \in X}\, setDist(x, S_{t-1})$
7: $\quad S_t \leftarrow S_{t-1} \cup \{x_{max}\}$
8: $\quad t \leftarrow t + 1$
9: **end while**
10: **return** S_t

have been proposed. Those algorithms can be broadly divided into two categories [31]: i) Greedy Algorithms (e.g., [18, 85, 97, 103]) and ii) Interchange Algorithms (e.g., [97, 99]).

Greedy Algorithms

Greedy algorithms are very popular for solving diversification problem as they are intuitive and efficient. It has been shown that the solution provided by the greedy algorithm is a $\frac{1}{2}$ -approximation of the optimal solution and that no polynomial algorithm can provide a better guarantee [91]). Generally, a Greedy algorithm starts with an empty set S and iteratively adds a result from X to S, based on some criterion. It terminates once k results are selected in S. An alternate approach is to initialize set S with all results in X and iteratively remove one result from S until only k results are left in S [40]. In most works, the former approach is used and is formally called *greedy construction algorithm* [31, 97].

In particular, Greedy construction algorithm initializes the diverse subset S by some result in X. Then, it proceeds through a number of iterations, until k results have been selected. The result that is selected in each iteration is the one that has the maximum set distance from S. The *set distance*, denoted $setDist(x, S)$, between a result x and a set S is derived directly from the definition of $f(S, d)$. Hence, for *MaxSum* diversity function the set distance is defined as:

$$setDist(x, S) = \frac{1}{|S|} \sum_{j=1}^{i} d(x, x_j)$$

and for *MaxMin* diversity function set distance is defined as:

$$setDist(x, S) = \min_{x_j \in S} d(x, x_j)$$

The details of greedy construction algorithm are presented in Algorithm 1. Many variations of greedy construction algorithm are used in literature. For instance, in [35] the initial diverse subset S is initialized by the most distant pair in X. Whereas in [40] S is initialized by a random result in X. In [42] another variation of greedy algorithm is employed. In each iteration, two results from the remaining results in X having maximum pairwise distance are added to the subset S. Often diversity is combined with some other ranking criterion, most commonly that of relevance to the user query. In that case, the initial subset S is initialized by the most relevant result in X [14]. In subsequent iterations, the best result maximizing the combined objective function is added to the subset S.

Interchange Algorithms

Like Greedy algorithms, interchange algorithms are also widely used in solving the diversification problem [63, 99]. In general, an interchange algorithm starts out with a complete initial solution and then attempts to find a better solution in the neighborhood of that initial one. For the diversification problem, the basic idea underlying the interchange algorithm is to start with an initial set S of size k and then iteratively modify the set S in order to improve the value of the diversity function. Generally, the subset S is initialized with k random results in X. In each iteration the result contributing least to the diversity of the set S is swapped with a result in X/S. This process continues until there is no possible interchanges that can improve the diversity of the set S. When combined with relevance, the initial subset S is constructed using most relevant k results [97]. The details of interchange algorithm are presented in Algorithm 2.

2.2.2 Complexity of Diversification Algorithms

The computational cost of Greedy construction algorithm, in terms of number of distance computations, is $O(k^2n)$ [31]. This asymptotic cost is the product of two components: the cost of calculating the set distance $setDist(x, S)$ and the number of times $setDist(x, S)$ is calculated. Let w_i denote the cost of calculating $setDist(x, S)$ at the i^{th} iteration, and t_k denote the number of such calculations required by the algorithm to identify the k diverse results. The number of iterations performed up to and including the i^{th} iteration can be modeled as a recurrence relation. In particular, at the first iteration (i.e., i = 1), the most diverse result can be trivially

Algorithm 2 Interchange.

Input: A set of query results X, an integer k.

Output: A set S with the k most diverse results in X.

```
1: Set S to be a random solution
2: while interchange occurs do
3:    find s_i, s_j s.t d(s_i, s_j) = min (d(s_i, s_j) : s_i, s_j ∈ S, s_i ≠ S_j
4:    for all x_i ∈ X\S do
5:       S' ← S\s_i ∪ x_i
6:       S'' ← S\s_j ∪ x_i
7:       if f(S', d) > f(S, d) and f(S', d) > f(S'', d) then
8:          S ← S'
9:       end if
10:      if f(S'', d) > f(S, d) and f(S'', d) > f(S', d) then
11:         S ← S''
12:      end if
13:   end for
14: end while
15: return S
```

selected as any random result in X, independently of the other diverse results, since $S = \Phi$ at this point. Thus t_1 is 0. At the i^{th} iteration, with $i > 1$, the set distance $setDist(x, S)$ is calculated for each result $x \in X/S$, which amounts to a total of n-(i-1) results. Therefore, t_i= n-(i-1). These two observations can be modeled as the base and recursion steps of a first-order linear recurrence, respectively [22]. It can be easily shown by solving this linear recursion that $t_k = O(kn)$ for $k < n$. Since, the cost w_i for calculating set distance is at most $k - 1$ in terms of distance computations, the total cost of Greedy construction algorithm is $O(k^2n)$.

The complexity of the Interchange algorithm depends upon the number of iterations and the cost incurred to compute diversity of set S in each iteration. The diversity of set S of size k can be computed in $O(k^2)$ time. However, the number of iterations for interchange algorithm are not known in advance. In the worst case, Interchange algorithm can iterate to try all possible combinations of k diverse subsets of set X of size n. Hence, in the worst case Interchange algorithm can perform $O(n^k)$ iterations.

2.3 Scalable Diversification for Data Exploration

As mentioned above, the diversification algorithms are based on the computation of pairwise comparisons. In general, $O(k^2n)$ comparisons are necessary for generating a diverse subset of size k out of n results. While this polynomial cost is still feasible in scenarios with small amounts of data with few user queries, it becomes prohibitive when computing diverse subsets of large data across numerous user queries. Hence, there is a need to revisit the diversification problem from the perspective of exploratory search. In particular, data exploration platforms pose following challenges to diversification of results in databases.

- *Scalability in Data*: For high dimensional large data sets, the diversification methods are expected to scale in both the database size, e.g. thousands of query results, as well as in the number of dimensions, e.g., tens to hundreds of the attributes in the database.

- *Interactive Performance*: Data exploration process involves processing of series of queries. The result of each query must be computed with interactive performance, e.g., within seconds, in order to minimize the overall exploration time. Hence, guaranteeing interactive performance when diversifying results of large number of queries becomes another key challenge.

To address the aforementioned challenges, novel solutions for processing diversification queries are needed that are scalable to data-intensive and time consuming exploration tasks. Typically, techniques like indexing, caching and multi query optimization are employed for efficient query processing in databases. Below, we summarize some of the existing diversification methods based on those techniques and discuss the applicability of those methods in data exploration.

- **Indexing:** In general, indexing techniques are used for efficient retrieval of data. The existing diversification solutions based on indexing employ distance based access methods. Those methods are generally based on pre-computation and indexing of the solution space considering one query spanning whole data set. For instance, a Cover tree based implementation of greedy algorithm is proposed for MaxMin diversification problem in [30]. In particular, diverse subsets for all possible values of k are indexed using a Cover tree. Similarly, a spatial index called M-tree is used in [32] for the fast computation of diverse results based on coverage based diversity measure. Indexing is also used in [39] to implement relevance and distance based sorted access methods to retrieve the top-k

diverse results. A space partitioning and probing algorithm is employed to minimize the number of accessed results, by pruning space around already selected results.

A pre-indexing and probing approach is used to retrieve a diverse results of queries with varying selection predicates and values of k in [96]. Only a bounded number of results within a distinct value are explored and a B+ tree index is used to skip over the similar results. However, the proposed methods consider diversity based on priority ordering of attributes. Two results that differ in a highly important attribute are considered highly diverse, even if they are similar in other low priority attributes. Such a diversity measure allows the exploitation of a Dewey encoding of results that can be used to build a B+ index. A similar problem is addressed in [62] in which a novel D-index is proposed that is based on the concept of computing a core cover, for evaluating both static as well as dynamic diversity queries. Indexing methods used in both [62, 96] are based on specific measure of diversity which is not applicable in the general case.

The application of index based diversification methods is limited in modern data exploration platforms. Firstly, the time required building state of the art indices over millions of data records can be a significant bottleneck in data exploration systems. Secondly, in the absence of predefined queries the indexes build a priori will rarely be useful [104]. For instance, the indexes used in [30, 32, 39] are build by computing distances between results along all data dimensions. Those indexes are no longer valid when new queries are posed that select subset of data projected along some of the dimensions. Hence, the fast computation of diverse results is gauranteed as long as the same query is repeatedly posed with varying values of k. The indexing methods used in [62, 96] require users to explicitly specify the priority ordering of each attribute which is often not feasible for users with little familiarity of underlying attribute space.

- **Data Caching:** Data caching is another effective technique for reducing query processing time in databases [27, 29]. Caches are used to both speed up repeated accesses to static data and to avoid recomputation by storing the results of a computation. For result diversification problem, caching has been mainly used for addressing the continuous diversity problem where diversified subsets are computed for each sliding window over data streams [33, 65]. Instead of re-evaluating all the k diverse results, the proposed scheme in [33] initializes the diverse subset of the new data window using the diverse results from the previous window. Whereas, in [65] an interchange algorithm is proposed to update the diverse subset as new data objects arrive. As a new object arrives it is replaced with

one of the existing objects in the current solution only if this replacement increases diversity. The cost of diversity evaluation of current solution is reduced by caching distance computations.

Multi-query optimizations: Multi query optimization techniques have been widely used to minimize the overall processing time of multiple queries [75]. However, most of the existing work on multi query optimization is focused on efficient query processing for fast retrieval of query results. For the post query processing tasks like diversification, only in [19] a multiple query diversification scheme is proposed in the context of microblogging. Specifically, the goal in [19] is to compute the smallest subset of posts that cover all other posts relevant to multiple queries with respect to a diversity dimension. The solution proposed in [19] only considers one dimension for diversity i.e., time or sentiment. While the single dimension diversity model is appropriate for the applications like microblogging, it is rarely applicable in database search where diverse representatives need to be computed along multiple dimensions.

Despite substantial work towards efficient methods of diversification, there has been limited focus on the scalable diversification methods specifically suited for data exploration. For diversification of high dimensional large data sets, schemes beyond traditional indexing methods are needed. Moreover, in a multi query data exploration environment, caching and multi-query optimization techniques need to be extended from query processing to post query processing tasks i.e., diversification. In particular, cache based diversification can provide the ability to compute diverse query results quickly based on the diverse results of prior queries that have been stored by the system. Similarly, multi query optimization techniques can be applied for efficient computation of multiple diverse subsets of multiple concurrent queries. Thus, to bridge the gap between current work on diversification and data exploration, in this thesis we have proposed novel diversification techniques that work in synergy with data exploration systems. In particular, we have presented diversification algorithms that are scalable to dimensionality of data, number of users, session length and complexity of diversity objective function.

Chapter 3

Progressive Diversification for Data Exploration Platforms

Data diversification plays an important role in providing users with a concise and meaningful representation of query results in exploratory search. Meanwhile, current techniques to data diversification follow a general *process-first-diversify-next* approach, in which a query is first executed on the database to generate a result, then the generated result is diversified to extract a small diverse set. One drawback of that approach is that all the data points (i.e., tuples) in a query result are accessed from disk while the user needs only few diverse results. Similarly, all the dimensional attributes of all the data points in a query result are considered for diversification. However, only very few of those points will be included in the diverse set, while most of the remaining points will be discarded. Clearly, that generic approach would hinder the performance of applying diversification in modern data exploration platforms that employ high-dimensional large databases. The result sets returned by those databases for user queries exhibit both high-dimensionality and high-cardinality.

Motivated by the need to efficiently provide users with diverse results during data exploration, here we propose the *Progressive Data Diversification (pDiverse)* scheme. The main idea underlying pDiverse is to utilize *partial distance* computation to reduce the amount of CPU and I/O incurred during query diversification. In a traditional approach where diversification is decoupled from query processing, our scheme allows to quickly detect and prune those data points in the query result that cannot be included in the final diverse set. The early pruning of those points provides significant reduction in the CPU cost required for distance computations.

Moreover, we propose integrating data diversification with query processing, which enables pushing down partial distance computation closer to the raw data, and hence provides pruning

at the data storage layer. This allows for saving the I/O costs incurred in accessing those pruned values, especially in vertically partitioned data (i.e., column-stores). In particular, *Column-store* systems vertically partition a database into a collection of individual columns, which are stored separately [1]. Hence, instead of reading all the attribute values of all the data points accessed by a query, pDiverse leverages a column-based storage to selectively read only those attribute values of the unpruned data points and save I/O cost. We summarize our contributions below:

- We propose the *Progressive Greedy (pGreedy)* heuristic, which forms the core of our pDiverse scheme. pGreedy utilizes partial distance computation for pruning and reducing CPU costs.
- We extend pGreedy to work in synergy with vertically partitioned data (i.e., column-store), which provides substantial reductions in I/O cost.
- We propose an integrated model, in which the processing of a range query is combined with the diversification of its results towards achieving further reductions in both CPU and I/O costs.
- We further optimize pDiverse by incorporating novel techniques for ordering of dimensions and approximation of diversity.
- We conduct extensive experimental evaluation on real and synthetic data sets, which illustrate the benefits achieved by pDiverse.

The rest of this chapter is structured as follows. We present query diversification model in Section 3.1. Next we introduce our pDiverse scheme in Sections 3.2 and present optimization techniques included in pDiverse in Section3.3. Our evaluation testbed and results are reported in Sections 3.4 and 3.5, respectively. We present the related work in 3.6 and we conclude in Section 3.7.

3.1 Query Diversification Model

We consider queries submitted to a database system or data exploration platform (e.g., [15, 54]). Such queries retrieve a number of results, or items, from the database. For instance, consider a database $\mathcal{DB}$, which consists of a set of D-dimensional tuples. Each tuple $x_i =< x_{i,1}, x_{i,2}, ..., x_{i,D} >$ is basically a point in a D-dimensional space, where the value of $x_{i,j}$ is drawn

from the domain of attribute A_j. Recall from chapter 2 that in a data exploration platform, a user typically explores a D-dimensional database by posing a sequence of range queries. A range query over databse $\mathcal{DB}$ is simply represented as a multi-dimensional box, also known as hyper-rectangle. Given a range query Q, let $X = \{x_1, \ldots, x_n\}$ be the set of data points that fall within the multi-dimensional range specified by Q. The aim of query diversification is to present a small diversified subset of X to the users instead of showing them all the results in X.

Below we present summary of the diversification problem detailed in Chapter 2 and revisit the complexity of Greedy algorithm.

3.1.1 Data Diversification

There are various definitions of diversity [97]. As mentioned in Chapter 2, we primarily focus on the widely used content-based definition of diversity in this thesis. Content diversity is an instance of the *p-dispersion* problem [36], whose objective is to maximize the overall dissimilarity within a set of selected objects. In particular, given a distance function $d(.,.)$, which measures the distance between two data points (e.g., Euclidean distance), the diversity of a set S is measured by a *diversity function* $f(S)$ that captures the dissimilarity between the results in S. Without loss of generality, in this work, we focus on measuring the diversity based on the average of the pairwise distances between results which is formulated as:

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

Putting it together, the data diversification problem is formally defined as follows: 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 function and $f(.)$ a diversity function. Then, the diversification problem is defined as selecting a subset S^* of X, such that:

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

Identifying an optimal diverse subset S^* has been shown to be NP-hard (e.g., [35]). Therefore, approximation methods are typically employed to select a near optimal diverse subset. Among many approximation methods, greedy-based heuristics are the ones most widely used because of their simplicity and performance. For instance, Greedy algorithm(Algorithm 1), initializes the diversified set S by selecting a random result in X. Then, it proceeds through

a number of iterations, until k results have been selected. In each iteration t, the result with the maximum set distance from the already selected results is added to the partially computed diverse set S_t. The *set distance*, denoted $setdist(x_i, S_t)$, between a point x_i and a set S_t is derived directly from the definition of $f(S)$ as:

$$setdist(x_i, S_t) = \frac{1}{|S_t|} \sum_{\substack{j=1 \\ x_j \in S}}^{t} d(x_i, x_j)$$

From the above, it is straightforward to derive the complexity of Greedy Construction in terms of number of distance computation tasks as $O(k^2 n)$ where $n = |X|$ [31]. However, caching distance computations across different iterations leads to further reduction in the number of pairwise distance computations, resulting in a complexity of $O(kn)$ [65]. A pairwise distance computation in a multi-dimensional space is typically performed under the L_p-norm metric. In particular, given any two D-dimensional data objects x_i and x_j, the L_p-norm distance function $d(x_i, x_j)$ is defined as: $d(x_i, x_j) = \sqrt[p]{\sum_{m=1}^{D}(x_{i,m} - x_{j,m})^p}$ where $1 \leq p \leq \infty$. Hence, irrespective of the choice of p, the complexity of Greedy in terms of number of CPU operations is $O(knD)$.

3.1.2 Problem Definition

Current data diversification techniques compute a diverse subset in two steps: 1) a query Q is executed on the database $\mathcal{DB}$ to generate a result X, and 2) the result S is diversified to generate a small diverse set S. Hence, in the first step, all the data points in X are read from disk while only few diversified results are needed by the user. Furthermore, in the second step, pairwise distance computations are applied to all the data points in X while most of those points will not be included in the diverse set S. Ideally, only those data points that belong to S should be accessed and presented to the user. While such an oracle solution is unrealistic, our goal in this work is to minimize the amount of accessed data (i.e., the I/O cost incurred in step 1) and the amount of distance computations (i.e., the CPU cost incurred in step 2), which are defined as follows:

Definition 5. Diversification I/O Cost $C_{I/O}$: *Given a query Q over database $\mathcal{DB}$, $C_{I/O}$ is the number of disk blocks accessed to produce a diverse subset S of size k for query Q.*

Definition 6. Diversification CPU Cost C_{CPU}: *Given a query Q over database $\mathcal{DB}$, C_{CPU} is the number of distance computations incurred to produce a diverse subset S of size k for query Q.*

3.2 Progressive Data Diversification Scheme

Distance computation is a core requirement of data diversification. However, calculating such pair-wise distance for all points in X across all D dimensions incurs high CPU costs, especially when only few points (i.e., k) are needed. Taking this observation further, the decoupling between processing Q and its diversification, incurs significant additional CPU and I/O costs for generating the results X, where again very few of those results are included in the final diverse set S. Motivated by those two observations, we present the pDiverse approach for data diversification. The main idea underlying pDiverse is to utilize partial distance computation to reduce the amount of CPU and I/O incurred during query diversification. Such goal can be achieved during the following two stages:

1. **Result Diversification:** utilizing partial distance computation allows to quickly detect and prune those points in X that cannot be included in the final diverse set S. This allows for saving the CPU cost incurred in computing the distance over those pruned values.

2. **Query Processing:** integrating data diversification with query processing enables pushing down partial distance computation closer to the raw data, and hence provides pruning at the data storage layer. This allows for saving the I/O costs incurred in accessing those pruned values as well as the CPU costs incurred in processing them.

To achieve the benefits outlined above, pDiverse approach utilizes several techniques, which are described in the following sections.

3.2.1 Computing Partial Distances

Clearly, computing the pairwise distances between all data points in a query result (i.e., X) incurs a high computational overhead. The amount of that overhead is further magnified when considering that only a small number of points (i.e., k) are eventually selected in the final diverse set S. This observation motivates us to utilize *partial distance* computation as a solution for reducing the overall cost incurred in identifying the points in a diverse set. The basic idea is to quickly *prune* a data point once it is clear enough from its partial distance that it cannot be included in the final diverse set.

While relying on partial distance computations is expected to provide significant savings in terms of CPU cost (i.e., number of computations), the amount of savings in terms of I/O cost (i.e., amount of accessed data) depends on the underlying data storage model. In a row-store,

Algorithm 3 pGreedy.

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: $t \leftarrow 1$
3: $S_t \leftarrow \{x\}$
4: $t \leftarrow t + 1$
5: **while** $t < k$ **do**
6: **if** Greedy **then**
7: $x_{max} \leftarrow \text{argmax}_{x_i \in X} \sum_{x_j \in S_{t-1}} d(x_i, x_j)$
8: **else if** pGreedy **then**
9: $x_{max} \leftarrow hPrune(S_{t-1}, X)$
10: **end if**
11: $S_t \leftarrow S_{t-1} \cup \{x_{max}\}$
12: $t \leftarrow t + 1$
13: **end while**
14: **return** S_t

for any data point x_i, which has been pruned after computing a partial distance on dimensions $A_1, ..., A_h$, where $h \ll D$, the amount of CPU savings is proportional to $D - h$. However, since x_i is stored as a row, all the attributes values of x_i are already read from storage leading to no savings in terms of I/O. To the contrary, column-store systems (e.g. C-store [89] and MonetDB [49]) vertically partition a database into a collection of individual columns, which are stored separately [1]. A column-store enables queries to read just the attributes they need, rather than having to read entire rows from disk and then discard those unneeded attributes after already incurring the I/O costs for reading them [44]. Thus, to complement the savings in CPU computation with corresponding savings in I/O access, pDiverse utilizes the benefits of the vertically partitioned storages (i.e., column-store).

3.2.2 Progressive Greedy Construction

Heuristics based on *Greedy Construction*, or Greedy for short, have been shown to provide both efficient and effective solutions to the data diversification problem [31, 97]. In this work, we propose *Progressive Greedy (pGreedy)*, which utilizes partial distance computation for efficient

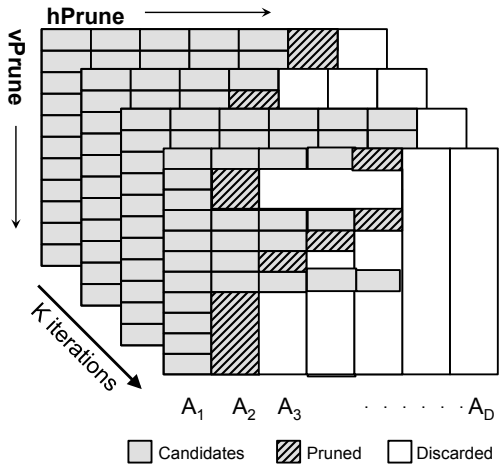


Figure 3.1: Horizontal and Vertical Pruning of candidate points

progressive data diversification, especially in high-dimensional databases. For clarity of presentation, in this section we assume that X is generated as the result of processing some query Q. Hence, all the data points in X are stored in memory and the goal is to generate a diverse set S while minimizing the number of distance computations (i.e., CPU cost). Such assumption is to be relaxed in later sections.

As shown in Algorithm 3, pGreedy follows the same steps as the original Greedy heuristic (e.g., [31, 97]), except for allowing pruning of values based on partial distances. Specifically, in each iteration t of pGreedy, a new point from X is selected to join the current diverse subset S_t. In the original Greedy heuristic, the distance of each point in X from S_t is calculated and the one with the maximum distance is chosen (step 6 of Algorithm 3). In pGreedy, however, that decision is based on partial distance computation, which allows for significant pruning, while providing the same exact solution achieved via full distance computation (step 8 of Algorithm 3).

To utilize partial distance, pGreedy relies on two operators: *hPrune*, and *vPrune* (shown in Algorithms 4 and 5, respectively), which work in tandem to enable efficient pruning of data values, as shown in Figure 3.1. Particularly, during each iteration of pGreedy, hPrune progresses incrementally through the dimensions (i.e., attributes) of the result X, allowing for some dimensions to be completely pruned from distance computation (i.e., *horizontal pruning*). If a dimension is not pruned, vPrune is invoked to process the data points in X under that dimension, while at the same time pruning some of those points (i.e., *vertical pruning*). Next, we first describe the details of vPrune, followed by hPrune.

Vertical Pruning The pGreedy heuristic relies on vPrune for eliminating data points that would naturally disqualify to join the diverse set S. Particularly, vPrune works at the dimension-level and it takes two parameters (Algorithm 4). The first parameter is a partial diverse subset

S_t of size t where $1 \leq t < k$. The second parameter is a list of candidate points $\mathcal{C}_h$, where each entry in $\mathcal{C}_h$ is defined as $(pid, < x_1, ..., x_h >)$ where $h \leq D$. That is, each entry in $\mathcal{C}_h$ is an incomplete h-dimensional point x for which the values of $D - h$ dimensions are unknown yet, whereas *pid* is a position identifier that simply points to the location of x in memory.

Essentially, vPrune prunes the points in $\mathcal{C}_h$ that cannot make it to S_t and returns a shorter list $\mathcal{C}'_h$. Particularly, $\mathcal{C}'_h$ contains the list of data points that survived the pruning, for which more dimensions need to be examined. Hence, vPrune is invoked again with an input $\mathcal{C}_{h+1}$, where $\mathcal{C}_{h+1} = \mathcal{C}'_h$.

In order to achieve that pruning, vPrune computes a partial set distance between each point in $\mathcal{C}_h$ and S_t. Without loss of generality, in this work, we focus on the widely used form of L_p-norm, in which $p = 2$ (i.e., Euclidean distance). To compute a partial Euclidean set distance, let X be a collection of D-dimensional points where $x = (x_1, x_2, \ldots, x_D)$ is a point in X. Hence, the Euclidean distance between two D-dimensional points x_i and x_j is defined as: $d(x_i, x_j) = \sqrt{\sum_{m=1}^{D}(x_{i,m} - x_{j,m})^2}$

For h-dimensional points, where $1 \leq h \leq D$, we define two distance bounds as follows:

Definition 7. Maximum distance bound ($d(x_i, x_j)^h_{max}$): *The maximum distance between two D-dimensional points x_i and x_j is the sum of two components: i) the distance between those two points in the h known dimensions, and ii) the maximum possible distance between those two points in the $D - h$ unknown dimensions. In a unit space, the latter component is equal to $\sqrt{\sum_{m=h+1}^{D}(1.0)^2}$. Hence, $d(x_i, x_j)^h_{max}$ is calculated as:*

$$d(x_i, x_j)^h_{max} = \sqrt{\sum_{m=1}^{h}(x_{i,m} - x_{j,m})^2 + \sum_{m=h+1}^{D}(1.0)^2}$$

Definition 8. Minimum distance bound ($d(x_i, x_j)^h_{min}$): *The minimum distance between two D-dimensional points x_i and x_j is calculated as:*

$$d(x_i, x_j)^h_{min} = \sqrt{\sum_{m=1}^{h}(x_{i,m} - x_{j,m})^2 + \sum_{m=h+1}^{D}(0.0)^2}$$

On the basis of the two distance bounds described above, we extend our previous definition of $setdist(.,.)$ (please refer to Section 6.1) to accommodate partial distance calculation. Particularly, we define a minimum set distance and maximum set distance of a point x from a diverse set S_t for h-dimensional points as follows:

$$minSetDist(x, S_t) = \sum_{x_i \in S_t}^{t} d(x, x_i)^h_{min}$$

Algorithm 4 vPrune.

Input: Partial Diverse Subset S_t, Candidate List of h-dimentional results $\mathcal{C}_h$

Output: Candidate List of h-dimensional results $\mathcal{C}'_h$

```
1: maxminDist ← 0
2: for all x_i ∈ C_h do
3:    mindist_{x_i} ← minSetDist(x_i, S_t, h)
4:    if mindist_{x_i} > maxminDist then
5:       maxminDist ← mindist_{x_i}
6:    end if
7: end for
8: while i < |C_h| do
9:    maxdist_{x_i} ← maxSetDist(x_i, S_t, h)
10:   if maxdist_{x_i} < maxminDist then
11:      C_h ← C_h − {x_j}
12:   end if
13: end while
14: return C_h
```

$$maxSetDist(x, S_t) = \sum_{x_i \in S_t}^{t} d(x, x_i)^h_{max}$$

For every entry x_i in the candidate list $\mathcal{C}_h$, vPrune computes the minimum and maximum set distance of point x_i from the current diverse set S_t as described above. If a point x_i has a maximum set distance less than the minimum set distance of any point x_j in $\mathcal{C}_h$ (i.e., maximum minimum distance), then x_i is immediately pruned. That is, if $maxSetDist(x_i, S_t) \leq maxminDist$, then x_i is pruned and is not considered any further because there exists another point in the candidate list which is guaranteed to provide better diversity than x_i. After processing dimension h, vPrune returns a shorter list of candidates to hPrune, which progresses to the next dimension (i.e., $h + 1$) as described next.

Horizontal Pruning

The work of vPrune, as described above, is orchestrated by hPrune (Algorithm 5). Particularly, hPrune progresses incrementally through the different dimensions and invokes vPrune to process those dimensions. During each iteration t of pGreedy, hPrune is initialized with the current diverse set S_t and it invokes vPrune to process the first dimension of all the points in X (i.e., $|\mathcal{C}_1| = |X|$). As it proceeds from one dimension to the next, the bounds imposed by

Algorithm 5 hPrune

Input: Partial Diverse Subset S_t, Result set X

Output: x_{max} with maximum set distance from S_t

1: $h \leftarrow 1$
2: **while** $h \leq D$ **do**
3: $\quad \mathcal{C}_h \leftarrow SELECT(C_h, Predicates_h)$
4: $\quad \mathcal{C}_h \leftarrow vPrune(S_t, \mathcal{C}_h)$
5: $\quad$ **if** $|\mathcal{C}_h|$=1 **then**
6: $\quad\quad x_{max} \leftarrow \mathcal{C}_h[0]$
7: $\quad\quad break$;
8: $\quad$ **else**
9: $\quad\quad h \leftarrow h + 1$
10: $\quad$ **end if**
11: **end while**
12: **return** x_{max}

	A_1 Max=1 Min=0	A_2 Max=0.7 Min=0.4	A_3 Max=0.8 Min=0.6	A_4 Max=0.6 Min=0.4
1	0.82	0.40	0.6	0.44
2	0.25	0.27	0.6	0.59
3	0.06	0.40	0.6	0.56
4	0.79	0.34	0.6	0.51
5	0.09	0.24	0.7	0.48
6	0.12	0.24	0.6	0.41
7	0.11	0.36	0.6	0.45
8	0.69	0.44	0.6	0.42

FIGURE 3.2: `SampleData` (accessed values are shaded and pruned values are blank)

partial distance computation grow tighter, and vPrune is able to eliminate more points as it processes more dimensions.

In each iteration t of pGreedy, hPrune terminates after it invokes vPrune on all dimensions, or when there is only one candidate point left in the candidate list $\mathcal{C}_h$. In the latter case, all $D - h$ dimensions that follow dimension h are not considered any further (i.e., horizontal pruning). In the subsequent iteration $t + 1$, the same process is repeated. Notice that some points that have been pruned in one iteration of pGreedy, might be re-considered in subsequent iterations. In particular, as a new point x_{max} is added to S in each iteration t, the distance between each point in X and the diverse set S needs to be updated to reflect the addition of that new point. Hence, points that have been pruned in iteration t might survive the pruning in one or more iteration $t' > t$.

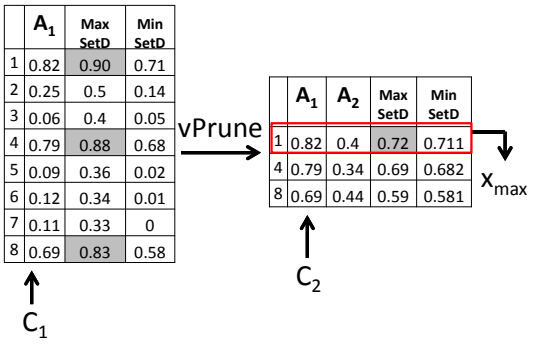


Figure 3.3: Selection of next diverse data point x_{max}

Illustrative Example Consider the data points in `SampleData`, as shown in Figure 3.2. Figure 3.3 shows an example of applying one iteration of pGreedy on those data points. In this example, assume that $t = 1$, hence S_t contains one data point. Further, assume that one data point to be $<0.11, 0.4, 0.7, 0.5>$ During processing attribute A_1, hPrune generates a candidate list $\mathcal{C}_1$, which contains a set of 1-dimensional data points. In turn, vPrune processes $\mathcal{C}_1$ and S_1 and computes maximum and minimum set distances for each point in $\mathcal{C}_1$. Notice that the maximum minimum set distance is provided by the first data point x_1 and is equal to 0.71. Hence all the points with maximum set distance less than or equal to 0.71 are pruned $(= x_2, x_3, x_5, x_6, x_7)$.

For the list of remaining points $(= x_1, x_4, x_8)$, attribute A_2 is accessed and a list of 2-dimensional data points $\mathcal{C}_2$ is created. Next, vPrune updates the partial set distances of the points in $\mathcal{C}_2$ and prunes all points with maximum set distance less than 0.711, which is the new maximum minimum set distance. Since only x_1 survived the pruning step, hPrune terminates and x_{max} is set to x_1 and added to S. In comparison to Greedy, where all the dimensions of all the data points are involved in distance calculations, pGreedy performs far less number of CPU operations for selecting each new point. For instance, in this particular example, hPrune computes distance across only 11 dimensional values as compared to 28 in case of Greedy.

3.2.3 Setting the Distance Bounds

Clearly, the power of pruning provided by pGreedy relies on the strength of the calculated minimum and maximum set distance bounds. That is, the tighter the bounds, the higher pruning power achieved by pGreedy. While setting the maximum distance between any two data points along an unknown dimension to 1.0 (as described in the previous section) guarantees the correctness of bounds, it often results in very loose bounds that reduce the pruning power.

In fact, a certain minimum number of dimensions h need to be fully accessed before achieving any pruning. In the case where that maximum is 1.0, that h is computed as follows.

Lemma 3.2.1. *In a D-dimensional unit space, given a point x and two h-dimensional points x_i and x_j where $h \leq D$. If the maximum distance limit along all D dimensions is 1.0, then $d(x_i, x)^h_{min} \geq d(x_j, x)^h_{max}$ is only be possible if $h \geq \frac{D}{2}$.*

Proof. The maximum distance bound between x_j and x is:

$$d(x_j, x)^h_{max} = \sqrt{\sum_{m=1}^{h}(x_{j,m} - x_m)^2 + \sum_{m=h+1}^{D}(1.0)^2}$$

Hence, the minimum value for $d(x_j, x)^h_{max}$ is achieved if the distance in all h dimensions is 0. That is, $\sum_{m=1}^{h}(x_{j,m} - x_m)^2 = 0$. Similarly, the minimum distance bound between x_i and x is given by:

$$d(x_i, x)^h_{min} = \sqrt{\sum_{m=1}^{h}(x_{i,m} - x_m)^2 + \sum_{m=h+1}^{D}(0.0)^2}$$

Hence, the maximum value for $d(x_i, x)^h_{min}$ is achieved if distance in all h dimensions is 1. That is $\sum_{m=1}^{h}(x_{i,m} - x_m)^2 = h$. Let's assume the minimum value for $d(x_j, x)^h_{max}$ and the maximum value for $d(x_i, x)^h_{min}$ then:

$$max(d(x_i, x)^h_{min}) \geq min(d(x_j, x)^h_{max})$$

$$= \sqrt{\sum_{m=1}^{h}(1)^2 + \sum_{m=h+1}^{D}(0.0)^2} \geq \sqrt{\sum_{m=1}^{h}(0)^2 + \sum_{m=h+1}^{D}(1.0)^2}$$

$$= h \geq \frac{D}{2}$$

□

Hence, it is required to access at least half of the dimensions before getting to prune any points based on their partial distances. In reality, however, the maximum achievable distance along any dimension is dependent on the domain range of that particular dimension. For instance, if the range of a dimension h is between 0.4 and 0.6 then the maximum distance between two points along this dimension cannot exceed 0.2. Hence, pGreedy utilizes the statistical information available from the database catalogue to formulate tighter bounds. Particularly, it simply uses the maximum and minimum value for each attribute A_i when computing maximum set distances. For instance, consider the first two points shown in Figure 3.2: $x_1 =< 0.82, 0.4, 0.6, 0.44 >$ and $x_2 =< 0.25, 0.27, 0.6, 0.59 >$. Further, assume that x_1 is the

only point in the diverse set S. If only the first attribute value of x_2 is available, then estimating the distance between x_2 and x_1 assuming a a full unit maximum distance will provide following distance: $d(x_1, x_2)^1_{max} = \sqrt{(0.57)^2 + 3 * (1)^2} = 1.82$. To the contrary, using the maximum and minimum values for each dimension (as shown in Figure 3.2), provides a tighter bound, which is calculated as: $d(x_1, x_2)^1_{max} = \sqrt{(0.57)^2 + (0.3)^2 + (0.2)^2 + (0.16)^2} = 0.69$.

In addition to leveraging the statistical information described above, pGreedy also automatically adjusts the upper bound on distance as the diverse subset evolves. This technique is based on the observation that as the size of diverse subset increases, the value of diversity decreases. This is due to the fact that diversification functions are *sub-modular* in nature [58]. Hence, the distance contribution of any new point cannot be larger than the contribution of the previous point added to set S. Accordingly, in each iteration, the current diversity value of the partial diverse subset S serves as the upper bound for the maximum set distance of any point.

3.2.4 Storage-aware Progressive Diversification

In the previous sections, our focus has been on reducing the computational costs incurred during data diversification by utilizing partial distances for enabling progressive diversification. The premise is that calculating pairwise distances for all points in X across all D dimensions is often unnecessary, which allows for efficient pruning, especially when only very few diverse points are needed (i.e., diverse set S). In this section, we further leverage progressive diversification to reduce the I/O costs required for generating a diverse set of points S. In particular, we migrate from the traditional *process-first-diversify-next* approach to an integrated approach, in which data diversification is weaved within query processing.

Recall that the basic idea underlying our proposed progressive diversification is to utilize partial distance computation to avoid computing pairwise distances along all of the attributes of a D-dimensional database. Depending on the data characteristics and diversification requirements, the pruning power can fall anywhere between pruning very few dimensions of very few data points, all the way to pruning entire dimensions for most of the data points. Such scenario naturally lends itself to take advantage of a vertically partitioned data layout, in which data is stored in columns instead of rows (i.e., column-stores).

Column-store systems vertically partition a database into a collection of individual columns, which are stored separately [1]. A Column-store enables queries to read just the attributes they

need, rather than having to read entire rows from disk and then discard those unneeded attributes after already incurring the I/O costs for reading them. Column-store systems employ concepts that are at a high level similar to vertical partitioning, in addition to many architectural features that are designed to maximize the performance on analytic workloads. One such feature is the use of virtual IDs for data access [49]. Particularly, one simple way to represent a column in a column-store involves associating a tuple identifier with every column (i.e., additional column). Alternatively, to reduce storage space, the position (i.e., offset) of the tuple in a column is used as a virtual identifier. Particularly, each attribute is stored as a fixed-width dense array and each record is stored in the same array position across all columns of a table. Relying on fixed-width columns simplifies locating a record based on its offset; for example accessing the i-th value in column A, simply requires to access the value at the location $base(A) + i \times width(A)$ [1].

Irrespective of the particular implementation of a column-store, our propose scheme pDiverse can leverage vertically partitioned data for reducing the incurred I/O costs. Particularly, instead of reading all the attribute values of all the data points in a result X then diversifying them, pDiverse can selectively read those attribute values for only the data points in the candidate list (i.e., unpruned points). For instance, consider a special case, in which hPrune (as described in Algorithm 5) narrows the candidate list to one single data point after progressively visiting h dimensions. In that case, hPrune terminates early and all the remaining $D - h$ dimensions are not read from disk.

While the case described above might be considered an extreme, during normal scenarios large savings in I/O cost remain attainable. Specifically, recall that in each iteration t of pGreedy, hPrune incrementally traverses through the different dimensions and prunes the candidate list of diverse points $\mathcal{C}$ accordingly. Hence, as it progresses from one dimension A_h to the next dimension A_{h+1}, more data points are pruned. Consequently, only the disk pages (i.e., blocks) from attribute A_{h+1} that contain the points in $\mathcal{C}_h$ are read from disk leading to progressive savings in I/O costs. Furthermore, as pGreedy moves to a next iteration $t + 1$, a disk page that contains the values of attribute A_h for some points in $\mathcal{C}_h$ is accessed directly from memory if it has already been read in some previous iteration. That is, only points in $\mathcal{C}_h$ that belong to uncached pages incur additional I/O costs.

In Section 3.3, we propose multiple optimization techniques that enable pDiverse to further extend the ideas above to minimize I/O costs. Meanwhile, in the next section we describe our method for integrating data diversification together with the processing of a range query (i.e., predicate evaluation).

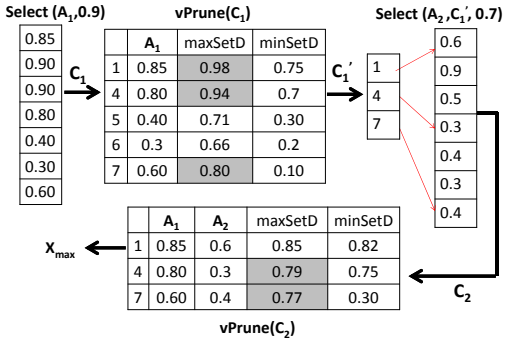


Figure 3.4: Selection of next divers result x_{max} for a range query.

3.2.5 Integrating Query Processing and Diversification

Users typically interact with a a multi-dimensional database by posing *range queries*. A range query Q is easily represented as a D-dimensional box (also known as hyper-rectangle), which is defined in terms of D range predicates $P_1, P_2, ..., P_D$. Each range predicate P_i is in the form $l_i \leq A_i \leq u_i$, where A_i is the i-th attribute (i.e., dimension), and l_i and u_i are the lower and upper limits specified by query Q along dimension A_i, respectively. Note that a dimension A_i that is not included in Q, is equivalent to $L_i \leq A_i \leq U_i$, where L_i and U_i are the lower and upper bounds of dimension A_i, respectively.

While current schemes adopt a two-stage approach to data diversification, in which a range query is executed first, then its result is diversified, we argue that an integrated approach is well positioned to minimize the costs of data diversification. In the previous section (Section 3.2.4), we highlighted the benefits obtained when diversifying the results of an unbounded range query. That is, a range query where each predicate P_i is defined as $L_i \leq A_i \leq U_i$. In this section, we address the general case of bounded range queries. In a nutshell, applying a predicate P_i on attribute A_i eliminates those data points that fail to satisfy the predicate. Similarly, applying hPrune on attribute A_i eliminates those data points that fail to join the diverse set. The combined impact of integrating those two operations results in a combined reduction in the number of data points that progress to the next attributes. Hence, reducing the amount of incurred I/O and CPU costs.

One straightforward approach towards achieving such integration is to encapsulate both range predicate evaluation together with distance pruning within hPrune. Particularly, when processing an attribute A_i, hPrune first invokes the database predicate evaluation operator (i.e., `select`), which returns a list of those points that satisfy the predicate P_i. That list is cascaded to vPrune to further eliminate the points that cannot make it to the diverse set.

For example, consider the execution of the following query as shown in Figure 3.4:

`SELECT` A_1, A_2, A_3

`FROM SampleData`

`WHERE` $A_1 < 0.9$ `AND` $A_2 < 0.7$

To answer that query, the select operator is applied to eliminate the values in column A_1 that that do not satisfy the predicate $A_1 < 0.9$ and $\mathcal{C}_1$ is created with points x_1, x_4, x_5, x_6, x_7. Then vPrune computes the maximum and minimum set distances for all candidates in $\mathcal{C}_1$ and only those points with maximum set distance greater than the minimum set distances of all the other points are kept in $\mathcal{C}'_1$ (i.e., x_1, x_4, x_7). The next select operator reads the values of the A_2 attribute for all the points in $\mathcal{C}'_1$ and returns the ones that satisfy the predicate $A_2 < 0.7$ (i.e., x_1, x_4, x_7). Finally, vPrune is invoked and points x_4 and x_7 are eliminated, leaving only point x_1 to be returned as the new diverse point x_{max}.

Though of its simplicity, the approach demonstrated above is expected to raise some interesting anomalies and provide an incorrect diverse set, when compared to a decoupled approach. To illustrate such anomaly, assume that the predicate condition on attribute A_2 is changed to $A_2 \leq 0.5$.

Figure 3.5 shows that as the select operator applies the new predicate on A_2, point x_1 is eliminated as it does not satisfy the predicate condition. Recall, however, that while pruning $\mathcal{C}_1$, vPrune eliminated points x_5 and x_6 based on the minimum set distance of x_1. Since x_1 is now filtered out, the pruning of x_5, x_6 becomes invalid and will cause an anomaly.

In order to avoid such anomaly, while at the same time utilizing the pruning power of partial distance calculation, pGreedy keeps track of the point with the maximum minimum set distance in $\mathcal{C}'_h$, let this point be x_m. In case x_m gets eliminated by the select operator, then some of the points pruned by vPrune in the previous step are revisited. Particularly, the next point x'_m with maximum minimum set distance is located in the list $\mathcal{C}_{h+1}$. Further, from the list of pruned points, we bring back those points that have maximum set distance higher than minimum set distance of x'_m. For example, Figure 3.5 shows that after eliminating x_1, point x_5 is added to $\mathcal{C}'_1$ since its maximum set distance is higher than minimum set distance of x_4, which is the new x'_m point.

3.3 pDiverse Optimizations

In this section, we extend pDiverse to include optimization techniques based on: i) ordering of dimensions, and ii) approximation of diversity. The details of those techniques are explained

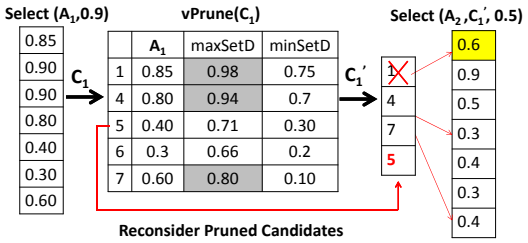


FIGURE 3.5: Handling range predicates

next.

3.3.1 Ordering of Dimensions

As presented so far, pDiverse perceives the different dimensions equally and hence, progressively processes them in the order determined by the scanner operator, which is based on selectivity. The selectivity of a predicate P_i over column A_i determines the number of data points that fall within that range (i.e., satisfy the predicate). Accordingly, in a column store, a column scanner operator consists of a series of pipelined scan nodes, where nodes that yield few qualifying tuples are pushed as deep as possible. That is, a column that produces fewer tuples is processed first so that to minimize the number of positions accessed in the next column and reduce processing costs.

Besides selectivity, imposing some additional criteria on the processing order of columns is expected to play an essential role in shaping the pruning power achieved by pDiverse. Particularly, in each iteration t of pGreedy, it is highly beneficial to prune as many points as possible and as early as possible. Hence, pDiverse visits columns according to selectivity, but for those columns with the same selectivity, it imposes a second level of ordering based on the following two factors:

1. **Attribute Domain (O):** Recall that as hPrune progresses from one dimension A_h to the next dimension A_{h+1}, it invokes vPrune, which updates the partial distances for all the points in the candidate list given the new values of attribute A_{h+1}. As those distances are updated, new bounds are set and more points are pruned from $\mathcal{C}_{h+1}$. However, if all the attribute values in dimension A_{h+1} are very close to each other, then updating the partial distances using those values will have negligible pruning power. To the contrary, if the attribute values in dimension A_{h+1} are spread over a fairly wide domain, then updating the bounds allows vPrune to easily distinguish the points that are still candidates for inclusion in S from those that are to be pruned. Hence, dimensions with wide value

domains should be processed earlier (i.e., assigned high priority), whereas processing dimensions with relatively smaller value domians should be postponed (i.e., assigned low priority).

2. **Attribute Size (Z):** It is expected that the number of pruned points increases, as hPrune progresses through the different dimensions. This is because as more dimensions are processed, the calculated bounds get tighter. Thus, to save I/O costs, it is intuitive to process those dimensions that are stored in fewer disk pages first. Hence, even if the entire dimension is to be processed, only very few disk pages are accessed. This is especially important given that in most column-stores, each attribute is stored as a fixed-width dense array. Accordingly, the values of a small-width attribute will be stored in smaller columns (i.e., fewer disk pages) than large-width attributes. For instance, a column of CHARs will take four times less I/O than a column of INTs.

Hence, each attribute A_i is assigned a score value based on its domain (O_i) and size (Z_i), which is defined as follows:

$$Score(A_i) = \frac{O_i}{Z_i}$$

Accordingly, dimensions are first ordered according to selectivity, whereas the scoring function above is used as a secondary criterion for breaking the ties among those dimensions which have the same selectivity.

3.3.2 Approximation of Diversity

The goal of pDiverse is to utilize partial distance computation to minimize the CPU and I/O costs required by data diversification. Meanwhile, pDiverse provides the same diversity (i.e., set of diverse points S) as that provided when using full distance calculations. However, like Greedy, pDiverse requires k iterations over the data to select k diverse results. Each iteration t is likely to explore some new points that have been pruned in previous iterations, thus reducing the benefits obtained from previous pruning. Hence, pDiverse incorporates simple approximation techniques that provide significant cost reduction, while incurring negligible loss in the diversity of the final result set. Our proposed approximation techniques are described next.

Page-based Pruning (PP): The main idea underlying this approximation is to give higher preference to prune those data points that require additional I/O accesses while still having low chance in joining the diverse set S. Particularly, recall that when selecting a new diverse point to be added to S, the candidate list of points $\mathcal{C}_h$ contains a set of h-dimensional points.

Next, vPrune processes list $\mathcal{C}_h$ to prune some data points and generate a shorter list $\mathcal{C}_{h+1}$. For each data point in $\mathcal{C}_{h+1}$, the attribute values of dimension A_{h+1} is accessed. For some points in $\mathcal{C}_h$, the value of the A_{h+1} dimension is already in memory from some previous iteration (i.e., in-memory point). Whereas for some other points, that value is on disk and has never been cached (i.e., on-disk point). That is, the disk pages that contain the attribute A_{h+1} values for those points have not been fetched in any previous iteration.

As mentioned in Section 3.2.2, the points in $\mathcal{C}_h$ that have maximum partial set distance less than the maximum minimum set distance of the point in $\mathcal{C}_h$ are pruned. However disk I/O cost can be reduced if pruning an on-disk point is given a higher preference as long as it is have minimum impact on the achieved diversity. To achieve this, let x_m be an in-memory point, and let its partial minimum set distance from set S be $minSetD(x_m, S)$. Further, let x_d be an on-disk point, and let partial its maximum set distance from set S be $maxSetD(x_d, S)$. Then if the following condition is met, x_d is marked as "to-prune" point: $\frac{(maxSetD(x_d,S)-(minSetD(x_m,S)}{(maxSetD(x_d,S)} \leq \theta$, where θ is a threshold parameter that determines the degree of approximation, Meanwhile, all the unpruned on-disk points that are added to the $\mathcal{C}_{h+1}$ list are marked as "to-read". In the end of vPrune, if a disk page that contains "to-read' points also contains some "to-prune" points, then those points are not pruned and are added to $\mathcal{C}_{h+1}$. To the contrary, if a disk page contains only "to-prune" points, then those points are pruned and that page is not read.

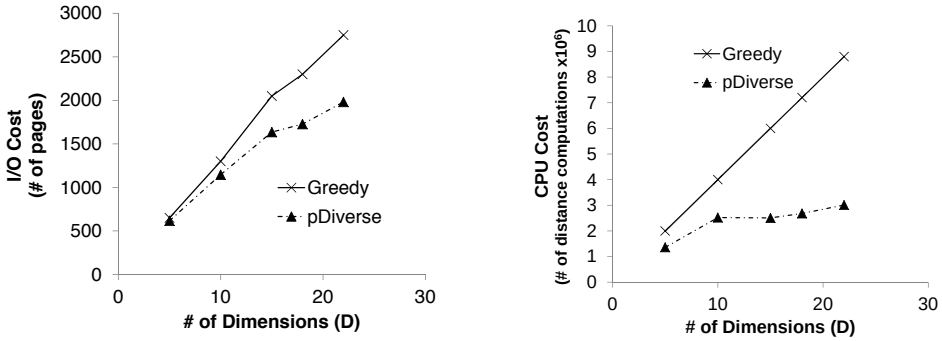
Column-based Pruning (CP): As mentioned above, the contribution of each dimension to the pruning power is proportional to its domain range. For instance, if all the attribute values in a certain dimension are very close to each other, then updating the partial distances using those values will have negligible pruning power. Hence, pruning those dimensions from partial distance evaluation is expected to provide substantial savings in terms of I/O costs, while incurring negligible effect on the achieved diversity. To utilize that observation, pDiverse employs a threshold value δ, such that if the normalized difference between the maximum and minimum values in a column A_h is less than δ, then dimension A_h is pruned and the column that stores the values of A_h is not accessed.

3.4 Experimental Testbed

We perform a number of experiments to evaluate the efficiency and the effectiveness of our pDiverse scheme. Table 6.3 summarizes the different parameters used in our experimental evaluation. **Schemes:** We evaluate the performance of the following schemes:

TABLE 3.1: Evaluation Setting.

Parameter	Range	Default
Number of Dimensions (D)	5–22	22
Database Size	100k	100k
Diverse Subset Size (k)	5–20	5
Page size (B)	4k–64k	4k
Page Pruning Threshold (θ)	0%–10%	2%
Column Pruning Threshold (δ)	0%–5%	1%



(a) I/O Cost (b) CPU Cost

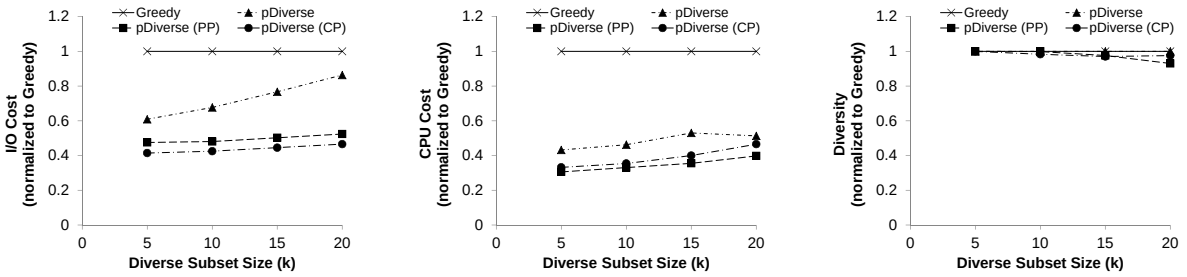
FIGURE 3.6: Impact of Number of dimensions

- Greedy: Retrieves all relevant query result from disk and then applies the Greedy Construction heuristic to select a diverse subset of query results.

- pDiverse: Progressively reads relevant query results and applies pGreedy to select a diverse subset. We evaluate various variations of pDiverse, which are described in the next section.

Datasets: We use both synthetic and real datasets. The dimensionality of our synthetic dataset varies in the range [5–22], with the default being 22-dimensional data. The values in each dimension are generated according to a zipf distribution, for which the skewness parameter is set in the range 0.0 (i.e., uniform) to 0.9 (skewed). Further, the width of each dimension is also generated according to a zipf distribution over the values: 2-bytes, 4-bytes, and 8-bytes. For all data sets, attribute values are normalized to [0–1]. Our Real dataset is generated from photoObj and SpacObj tables of Sloan Digital Sky Server Database. That data set contains 100k tuples and has 15 numerical dimensions.

Performance: We have implemented all the components of pDiverse from scratch in C++. Similar to [10, 44], we have implemented a simple vertically partitioned data store, in which



(a) I/O Cost (b) CPU Cost (c) Diversity

Figure 3.7: Impact of approximations

we experimented with different configurations of page size for storing data columns. Moreover, all reported performance results are for cold cache execution and prefetching degree of 0. The performance of each scheme is measured based on the following metrics:

- I/O Cost: measured as the number of data pages read from disk.
- CPU Cost: measured as the number of distance and comparison operations.
- Diversity: measured as the value of the diversity function of the diversified subset S.

Queries: Our query workload contains both bounded and unbounded range queries. An unbounded range query is basically extracting a diverse subset from all the data points in the database without applying any predicates, whereas a bounded range query specifies range predicates on some dimensions.

3.5 Experimental Evaluation

In Section 3.5.1, we present our evaluation results for simple unbounded range queries on real and synthetic databases, whereas our results for bounded range queries are presented in Section 3.5.2. Experimenting with unbounded range queries highlights the pure benefits of utilizing partial distance computation, independently from the additional benefits obtained from integrating predicate evaluation with data diversification.

3.5.1 Unbounded Queries

Impact of number of dimensions:

Figure 3.6(a) and Figure 3.6(b) show that both I/O and CPU costs increase with increasing the number of dimensions for both Greedy and pDiverse. However, pDiverse performs less number of I/O and CPU operations as compared to Greedy. As shown in Figure 3.6(a), the

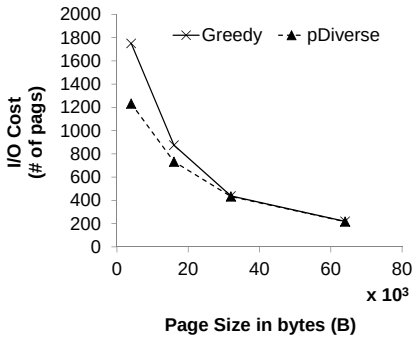
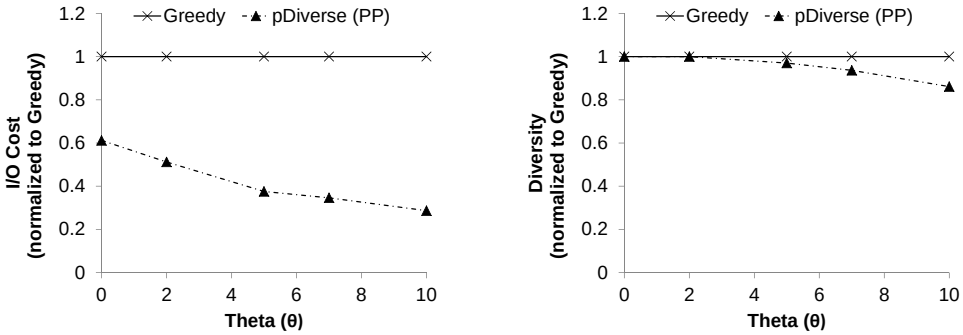


FIGURE 3.8: Impact of page size on I/O cost.



(a) I/O Cost

(b) Diversity

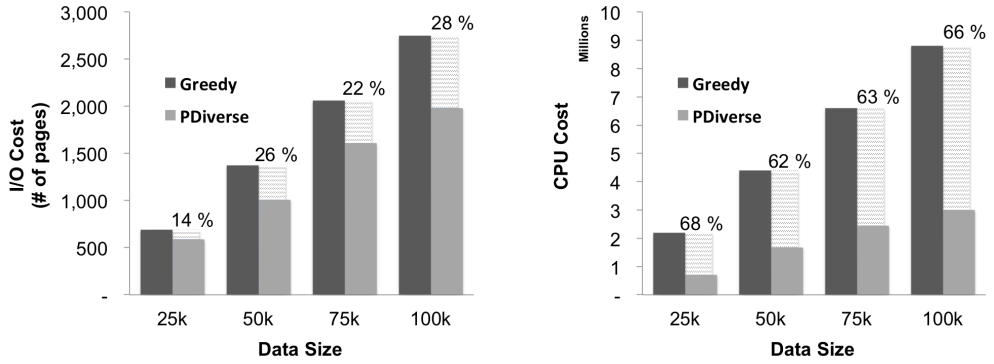
FIGURE 3.9: Impact of Theta (θ)

savings in I/O increase from 6% to 28% as the dimensionality increases from 5 to 22. Meanwhile, the savings in CPU operations increase from 32% to 66% as shown in Figure 3.6(b).

Impact of page size: Figure 3.8 shows the I/O cost incurred by pDiverse with increasing page size. As the figure shows, for page sizes of 4k and 16k, pDiverse manages to reduce I/O cost by 30% and 17%, respectively. However, if the page size is very large, many column values are packed in a single page. Thus even if only few values are accessed in a page, the whole page is still read from disk and reductions in I/O cost is minimal. This highlights the need for our I/O-based approximations presented in Section 3.3 and evaluated next.

Impact of Data Size:

In this experiment, we have evaluated the scalability of our proposed scheme to data size. As shown in Figure 3.10, the number of CPU and I/O operations increase for both Greedy and pDiverse as the data size increases. However, pDiverse consistently performs much less number of CPU and I/O operations as compared to Greedy. Figure 3.10(a) shows that pDiverse performs on average 22% less I/O operations for datasets of various sizes. Similarly, in Figure 3.10(b), it has been shown that pDiverse performs around 64% less CPU operations as the data set size varies from 25000 tuples to 100,000 tuples. As the pruning power of pDiverse is



(a) I/O Cost (b) CPU Cost

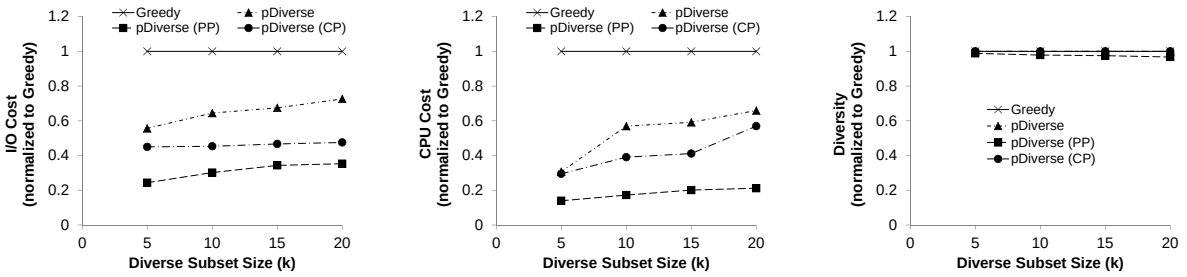
Figure 3.10: Impact of data size on I/O and CPU cost.

independent of the number of tuples in a data set, we do not see an increasing or decreasing trend in the cost savings for pDiverse as the data set size varies.

Impact of approximations: In this experiment, we evaluate the performance of our approximation methods, namely: pDiverse(PP) and pDiverse(CP), which apply page- and column-based pruning, respectively. Recall that pDiverse(PP) is controlled using a threshold parameter θ, whereas pDiverse(CP) is controlled using a threshold parameter δ. In this experiment, we use the default values $\theta = 2\%$, and $\delta = 1\%$. Figure 3.7(a) shows the performance in terms of I/O with increasing the value of k. As the figure shows, both approximation methods pDiverse(PP) and pDiverse(CP) perform better than the original pDiverse. Figure 3.7(b) shows similar trend for CPU operations. While pDiverse(CP) incurs the least amount of I/O since few columns are completely pruned, pDiverse(PP) performs better than pDiverse(CP) in terms of CPU operations for higher values of k. This is because pDiverse(PP) is able to prune more values vertically.

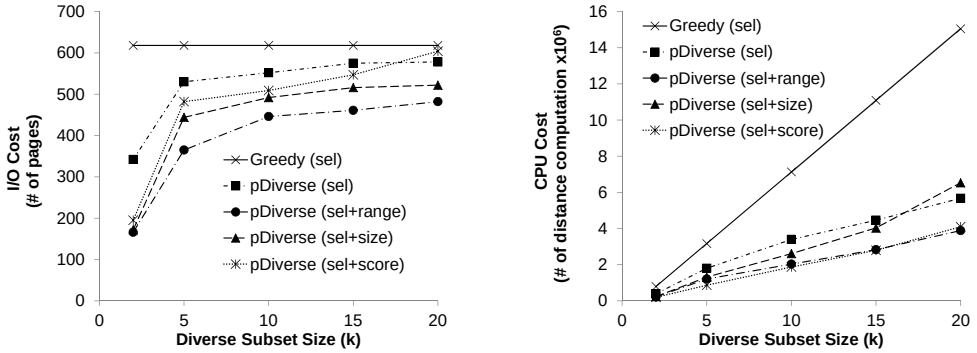
Figure 3.7(c) shows that approximation methods are able to achieve diverse subsets of comparable diversity to Greedy and pDiverse. However, some loss in diversity is seen as k approaches 20. The loss in diversity is higher for pDiverse(PP) as compared to pDiverse(CP). This is because pDiverse(PP) selects slightly less diverse in-memory points to avoid disk I/O, whereas the dimensions eliminated by pDiverse(CP) are those that are expected to have minimal impact on distance between two points.

Figure 3.9(a) shows the impact of θ on I/O costs. As the value of θ increases from 0% to 10%, the savings in I/O increase from 38% to 70%. This increase in I/O savings, however, comes at the expense of some loss in diversity, as shown in Figure 3.9(b). For instance, the diversity decreases by only 0.01% at $\theta = 5\%$, whereas as θ approaches 10%, the loss in diversity reaches 14%.



(a) I/O Cost (b) CPU Cost (c) Diversity

FIGURE 3.11: Performance of pDiverse on SDSS Dataset



(a) I/O Cost (b) CPU Cost

FIGURE 3.12: Impact of k on cost of Range Queries

SDSS real dataset: Figure 3.11 shows a similar performance of pDiverse and its approximation methods on the real data set generated from the SDSS database. For instance, Figure 3.11(a) shows that pDiverse is able to reduce I/O cost by more than 30%, whereas Figure 3.11(b) shows reductions in CPU cost up to 35%. However, by incorporating our proposed approximations, pDiverse is able to achieve around 65% savings in I/O cost and up to 80% savings in CPU cost.

3.5.2 Bounded Range Queries

In the following, we generated bounded range queries with range predicates posed on up to 4 out of the 22 dimensions. Range predicates are generated uniformly in the domain [0–1]. In order to assess the impact of our methods for ordering of dimensions, described in Section 3.3, we evaluated the following variants of pDiverse : i) pDiverse(sel): dimensions are ordered in descending order of their selectivity, ii) pDiverse(sel+size): dimensions are ordered in descending order of their selectivity, and the size of each dimension is used for tie breaking, iii) pDiverse(sel+range): dimensions are ordered in descending order of their selectivity, and the range of each dimension is used for tie breaking, and iv) pDiverse(sel+score): dimensions

are sorted in descending order of their selectivity and the score of each dimension, as computed in Section 3.3, is used for tie breaking.

Impact of diverse set size: Figure 3.12 shows the sensitivity of all the above schemes to increasing k. Naturally, Greedy is indifferent to the value of k as it provides no I/O optimizations, whereas the I/O cost incurred by pDiverse increases as k increases. Figure 3.12 shows that the savings in terms of I/O cost achieved by pDiverse are more than 30% for smaller values of k but decrease to around 7% for higher values of k. However, these savings are significantly higher for pDiverse variants, which employ methods for ordering of dimensions (i.e., pDiverse(sel+range), pDiverse(sel+size), pDiverse(sel+score)). For instance, for $k = 20$, I/O cost savings are up to 23% as compared to 7% achieved by pDiverse. It is worth highlighting that among the different variants of pDiverse, pDiverse(sel+score) performs better in terms of I/O since dimensions with smaller size and larger range are read first, which provides significant data reduction as more candidates are pruned earlier.

Figure 3.12(b) shows the CPU operations performed by each of the pDiverse variants in comparison to that performed by Greedy. Similar to I/O cost, all the pDiverse variants perform less number of CPU operations as compared to Greedy for all values of k. However, pDiverse(sel+size) performs higher number of CPU operations as compared to the other variants. This is because it prefers reading dimensions with smaller size first to save on disk I/O cost, without considering the pruning power of each dimension.

3.5.3 Summary of Experimental Evaluation

In our experimental study, we have evaluated the robustness of our proposed schemes to high dimensional datasets. The experiments conducted on both real and synthetic data sets measure the pruning efficiency and performance of various pDiverse variants. As compared to the baseline Greedy construction heuristic, our proposed pDiverse scheme performs less number of I/O and CPU operations for computing a diverse subset of same diversity as computed by Greedy. Those cost savings increase with the increase in the number of dimensions as the pruning power of pDiverse rely on the number of dimensions. Without using any approximations and reordering of dimensions, pDiverse performs up to 28% less I/O operations and 66% less CPU operations for a dataset with 22 dimensions. We have also evaluated the scalability of pDiverse to the diverse subset size k. For smaller values of k, pDiverse saves substantial I/O cost. However, for the higher values of k the I/O cost savings are decreased to only 7%. Hence, for higher values of k the variants of pDiverse with dimension ordering and approximations can

be used for better I/O performance. The pDiverse variants with approximations rely on tuning parameters for balancing the trade-off between the diversity of the subset and the computational cost. The I/O cost savings by approximate pDiverse variants are up to 40% for a negligible loss in diversity, which is equal to 0.01%.

3.6 Current Approaches to Efficient Diverse Set Selection

Efficient diversification schemes have been proposed in the literature to compute a diverse subset without considering all relevant results. For instance, in [62,96] pre-indexing and probing approaches are used to retrieve top-k diverse results. Database tuples are organized in a tree structure using a Dewey encoding which is later used to select the k most diverse tuples. However, the approaches presented in [62, 96] consider diversity based on priority ordering of attributes and are not applicable to general content based diversity measure. In contrast, pDiverse proposed in this chapter is generic to any L_p norm based diversity measure.

In [33], a cover tree based approach is used to index the data tuples with respect to their distances among themselves. Once a cover tree is constructed over a result set, it can be exploited to retrieve k diverse tuples in $O(k)$ time. Since, in [33] cover tree is used for dynamic diversification problem it is assumed that the user query remains same while underlying data is changing. Hence, a cover tree is initially generated using query result set and is then dynamically updated as the data changes. However, in our work we address the problem of executing different range queries over static data. In that case, constructing a new cover tree for every range query is infeasible. Another approach to generate diverse result set without retrieving all relevant results is proposed in [39]. A space partitioning and probing algorithm is used to minimize the number of accessed results by pruning space around already selected results. Unlike pDiverse, the proposed solution in [39] makes use of both relevance based and distance based sorted access methods to compute the diverse top k results. In our solution, we do not make use of any prior distance or relevance scores.

One of the key components of our proposed solution is computation of partial distances. Relying on partial distances has been shown to provide significant benefits in the context of k-NN similarity search (e.g., [10, 23, 59, 64]). However, to the best of our knowledge, this is the first work to investigate utilizing partial distance computation for data diversification. In the following, we highlight some of the substantial differences between the two problems. In k-NN,

distance comparison is performed with respect to a single query point, hence, k nearest neighbors can be selected within a single iteration over a candidate set, such that pruned points are never re-visited (as in [10, 23]). To the contrary, in data diversification, distance comparison is performed with respect to the current set of diverse points, hence, k iterations are needed, such that in the end of each iteration, a new data point is added to the diverse set. Accordingly, as the diverse set S evolves, points that have been previously pruned, might be re-considered again in future iterations. That dependency between the evolving set S and the data points in X challenges the pruning power of partial distance computation, and pDiverse has addressed that challenge.

3.7 Summary

In this chapter we presented the *progressive data diversification* scheme (pDiverse), which utilizes partial distance computation for pruning and reducing the CPU costs of diversification. We extended our scheme to work in synergy with vertically partitioned data (i.e., column-store), which provides substantial reductions in I/O cost. Our proposed integrated model combines the diversification of query results with the processing of a range query and hence achieves further reductions in both CPU and I/O costs. We have provided extensive experimental evaluation on real and synthetic data sets, which illustrate the benefits achieved by pDiverse.

Chapter 4

Concurrent Diversification of Multiple Search Results

Current Diversification schemes are centered around processing results of one query at a time. Scalable Diversification methods in general, attempt to minimize the processing cost of a single query (e.g., [39, 62, 96, 104]). Towards that end, in Chapter 3 we proposed the progressive diversification scheme for the efficient diversification of a single query result. However, Data exploration platforms host multiple users running multiple data analysis sessions. Many of those sessions are executed in parallel to maximize the utilization of resources. This leads to simultaneous execution of many exploratory queries. Such environments highlight the need for a diversification system that is able to effectively and efficiently diversify the results of many queries concurrently.

In this chapter, we present the *DivM* (Diversification of Multiple Search Results) scheme that targets the problem of efficiently diversifying the results of multiple queries. Towards this goal, DivM leverages the natural overlap in search results in conjunction with the concurrent diversification of those overlapping results. This enables DivM to provide the same quality of diversification as that of the sequential methods, while significantly reducing the processing costs. Moreover, DivM also exploits various approximation techniques that provide orders of magnitude reductions in processing cost, while maintaining a quality of diversification comparable to that of near optimal schemes. To achieve that, DivM incorporates methods for tuning the degree of approximation as well as refining the accuracy of the diversified results.

The contributions of this work are summarized below:

- We define the problem of Concurrent Diversification of Multiple search results
- We propose the *DivM* scheme that leverages the natural overlap in search results in

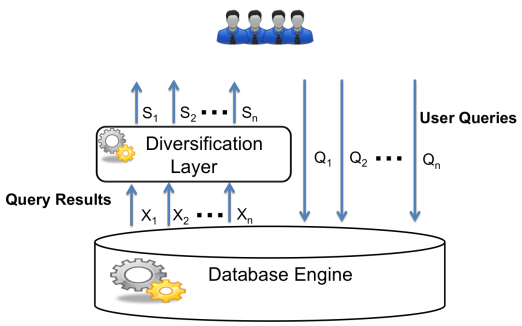


FIGURE 4.1: Multiple Search Result Diversification Model.

conjunction with the concurrent diversification of those overlapping results.

- We further generalize and extend the DivM scheme to exploit various approximation techniques for further reductions in diversification cost.

- We conduct extensive experimental evaluation on both real and synthetic data sets to shows the scalability exhibited by our proposed scheme under various workload settings.

The remainder of this chapter is organized as follows. We formalize the multiple search result diversification problem in Section 4.1. The *DivM* scheme and our algorithms are presented in Section 4.2. The experimental testbed and evaluation results are reported in Section 4.3 and Section 4.4, respectively. We conclude in Section 4.5.

4.1 Multiple Search Results Diversification

Since their early stages, data management platforms have been designed to support the concurrent execution of queries and transactions. Concurrency is an indispensable feature for any database system as it allows for maximizing the utilization of limited resources as well as exploiting the overlaps that naturally occur between the different data processing tasks. Supporting concurrency in database systems has motivated a wealth of well-studied research, especially in the areas of transaction management and multiple query optimization [73]. For instance, multiple query optimization techniques typically employ schemes that leverage partial aggregation [43] and data subsumption [75] in order to minimize the overall costs for processing multiple queries simultaneously.

This has been particularly beneficial in data exploration systems, in which multiple users are running multiple data analysis sessions (e.g., [15, 54, 102]). From a system perspective, this leads to the simultaneous execution of large number of queries that are essentially of exploratory

nature. For example, Figure 4.1 shows the interaction of multiple users with the database system. The parallel execution of user queries retrieve query results that are available at the same time. Those query results are then further processed to extract diverse subsets. Although, individual queries may be regarded as independent and generated by independent users. Often those queries may be similar or share the subspace of data. Under these circumstances, diversifying the set of overlapping results of various queries together is clearly beneficial. Hence, we formulate the problem of concurrent diversification of multiple query results as presented next.

4.1.1 Problem Definition

In environments where multiple queries are submitted by a number of different users, we define the diversification of multiple search results as follows:

Definition 9. *Let $\mathbb{Q} = \{Q_1, Q_2, \ldots, Q_N\}$ be a set of N user queries. Let X_i be the set of results that satisfy Q_i and k be a positive integer with $k \leq |X_i|$, $1 \leq i \leq N$. Let also d be a distance metric and f a diversity function. Then,* Multiple Search Result Diversification *is defined as selecting a set $\mathbb{S}^*$ of n subsets $\{S_1^*, S_2^*, \ldots, S_N^*\}$, such that:*

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

Without loss of generality, in this work we focus on the MaxSum diversity function defined as:

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

According to Definition 9, a diversification system should ideally locate an optimal set of diverse subsets $\mathbb{S}^*$ for the input queries. However, due to the inherit NP-hardness of the diversification problem, this is not feasible. Hence, to measure the efficiency and the effectiveness of a diversification system, we define the following metrics:

Definition 10. *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 a session of N queries is: $\frac{1}{N} \sum_{i=1}^{N} C(S_i)$.*

Definition 11. *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: $\frac{1}{N} \sum_{i=1}^{N} D(S_i)$.*

Hence, the success of any diversification system can be easily measured in terms of average diversification cost and average diversity. In particular, our goal in this work is to optimize the diversification of multiple query results (i.e., minimize diversification cost of computing each diverse subset S_i) while computing high quality diverse subsets (i.e., maximize diversity for each diverse subset S_i). In the next section, we present our scheme and discuss its impact on both the efficiency and effectiveness of diversification.

4.2 Concurrent Diversification of Multiple Search Results

Before presenting our DivM scheme, we first examine Naïve Greedy, a straightforward extension of the classical Greedy algorithm, in which multiple results are diversified sequentially and independently. Next, we present details of DivM scheme and explain its underlying fundamental ideas.

4.2.1 Naïve Greedy

A naïve solution to simultaneously diversifying multiple search results would be to retrieve the search result X_i for each query $Q_i \in \mathbb{Q}$ and then apply the Greedy heuristic (i.e., Algorithm 1) on each X_i separately to detect its respective diverse subset S_i. This method is called "Naïve Greedy".

Under this baseline approach, each search result is diversified independently and, thus, the complexity of Naïve Greedy is simply computed as: $O(k^2|X_1|) + \ldots + O(k^2|X_N|)$. Hence, the total processing cost of Naïve Greedy is:

$$C_{\text{Naïve-Greedy}}(\mathbb{S}) = O(Nk^2 \max_i |X_i|)$$

Naïve Greedy treats each user query independently, hence its complexity essentially increases linearly with the increase in the number of result sets to be diversified (i.e., N). In many real-life applications, however, it is often the case for many queries to have overlapping result sets.

Example 2. *Consider two queries submitted simultaneously to the Sloan Digital Sky Survey database. The first query, Q_1, retrieves all galaxies that are brighter than magnitude 22, given that local extinction is larger than 0.75. The second query, Q_2, retrieves all galaxies that are brighter than magnitude 18, given that local extinction is larger than 0.85. Both queries will retrieve all available results concerning galaxies brighter than magnitude 22 where the local*

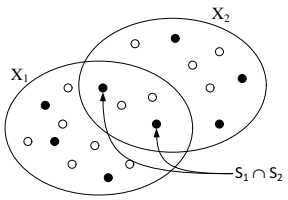


Figure 4.2: Result and diverse sets for two queries. Diverse items (i.e., results) are shown in bold.

extinction is greater than 0.85*. Depending on the available data, there may be significant data overlap between the two result sets* X_1 *and* X_2*. This data overlap might potentially translate into further overlap between the two diverse subsets* S_1*,* S_2 *selected for the two queries.*

Our proposed DivM scheme, described next, attempts to leverage this overlap among the result sets of the various queries for the efficient evaluation of their respective diverse subsets.

4.2.2 DivM

In comparison to the Naïve Greedy approach, instead of processing the overlapping portions of the results multiple times, DivM processes those portions only once leading to an overall amortized processing cost. Before explaining the details of DivM (in Sections 4.2.2 and 4.2.2), in the following we give an overview of DivM and explain its underlying fundamental ideas.

In principle, DivM can be perceived as an instance of the partial aggregation technique typically used in multiple query optimization (e.g., [43]). Result diversification, however, introduces further complexities due to the dependency between each result set (i.e., X_i) and its respective diverse set (i.e., S_i) during the computation of distance functions. In particular, result diversification is an iterative process, in which a raw result set X_i is continuously processed in conjunction with its respective partial diverse set S_i until the final diversity goal is achieved. Handling such dependency, while at the same time exploiting the opportunities of shared processing of overlapping data processing, is one of the features provided by DivM.

At a high-level, DivM is based on the Naïve Greedy algorithm presented in Section 4.2.1. In DivM, however, all the search results are processed concurrently and in each iteration, one item (i.e., result) of X_i is selected for inclusion in S_i, $1 \leq i \leq N$. Towards exploiting the overlap in search results for concurrent processing, the following two observations are made:

1. the distance function d between any item $x_j \in X_i$ and the items in S_i is computed independently of any other items in X_i, and

2. the distance function d between any item $x_j \in X_i$ and the items in S_i can be assembled from its partial values.

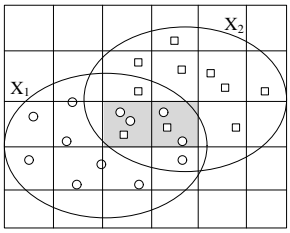


FIGURE 4.3: Mapping objects to the grid. Circles and squares denote results for Q_1 and Q_2 respectively. The shaded cells denote the overlapping region for the two queries.

To illustrate the second observation above, consider the following example:

Example 3. *Assume two queries Q_1 and Q_2 and their respective result sets X_1 and X_2. Further, assume that in any arbitrary iteration of DivM, S_1 and S_2 are the current diverse sets of X_1 and X_2, respectively. In such iteration, S_1 and S_2 might have few overlapping results (as shown in Figure 4.2). In that case, S_1 can be clearly expressed as the difference between S_1 and S_2 union their intersection. That is, $S_1 = ((S_1 \backslash S_2) \cup (S_1 \cap S_2))$. S_2 can be expressed similarly.*

In the example above, for a single item $x_j \in X_1$, the distance function $d(x_j, S_1)$ can be computed as:

$$d(x_j, S_1) = d(x_j, S_1 \backslash S_2) + d(x_j, S_1 \cap S_2)$$

Similarly, if item $x_j \in X_2$, the distance function $d(x_j, S_2)$ can be computed as:

$$d(x_j, S_2) = d(x_j, S_2 \backslash S_1) + d(x_j, S_1 \cap S_2)$$

Hence the term $d(x_j, S_1 \cap S_2)$ can be evaluated only once for every item $x_j \in X_1 \cap X_2$ when computing $d(x_j, S_1)$ and $d(x_j, S_2)$.

Clearly, the calculation of $d(x_j, S_1)$ outlined above is an example of applying partial aggregation, in which the final value of the distance is easily assembled from its partial values. This is applicable over all distributive and algebraic distance functions that are typically used in measuring (dis)similarity, such as all variants of L_p norm including the Euclidean distance.

The combination of the first and second observation listed above allows DivM to exploit the data overlap exhibited by the queries in Example 3 along two orthogonal dimensions:

1. **Overlap in Result Sets:** Process the sets $X_1 \backslash X_2$, $X_2 \backslash X_1$ and $X_1 \cap X_2$ separately at each iteration of the algorithm, and

2. **Overlap in Diverse Sets:** Process the set $S_1 \cap S_2$ only once at each iteration of the algorithm.

Towards detecting and leveraging overlaps in both the result sets and diverse sets, DivM incorporates a *preprocessing* phase in which:

1. **Grid Construction:** DivM imposes a multi-dimensional *grid* structure on the available results space [67]. In particular, this grid structure is used to partition the results space into D-dimensional cells, where D is the dimensionality of the search results. For simplicity, we assume that all dimensions are normalized and are in the range $[0, 1]$. The grid cells are obtained by partitioning each dimension into intervals of equal width γ. That is, γ is the *resolution* of the grid.

Algorithm 6 DivM-Point

Input: Result sets (X_1, X_2...X_N), an integer k.

Output: S_1, S_2...S_N with the k most diverse items of (X_1, X_2...X_N) respectively.

$X \leftarrow X_1 \cup X_2 \ldots \cup X_N$
for all Queries **do**
  $q_i.S \leftarrow$ random x where $x \in X_i$
end for
while $|q.S| < k$ **do**
  for all $x_i \in X$ **do**
    $Q \leftarrow$ all queries sharing x_i
    $S' \leftarrow q_1.S \cap q_2.S \ldots \cap q_{|Q|}.S$
    $d_x \leftarrow d(x_i, S')$
    for all $q_i \in Q$ **do**
      $d_s \leftarrow d_x + d(x_i, q.S \backslash S')$
      if $(d_s > d(q.candidate, q.S)$ **then**
        $q.candidate \leftarrow x_i$
      end if
    end for
  end for
  for all Queries **do**
    $q_i.S \leftarrow q_i.S \cup \{q_i.candidate\}$
  end for
end while
return $q_1.S, q_2.S, \ldots, q_N.S$

2. **Data Mapping:** DivM scans the set of multiple results available for diversification (i.e., X), each result (i.e., item) $x_j \in X_i$ is perceived as a point in the partitioned multi-dimensional space created by the grid structure. Hence, for each point x_j, DivM locates a

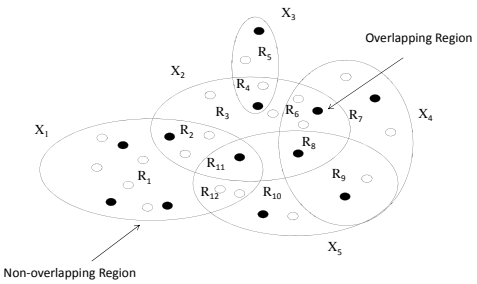


FIGURE 4.4: Result and diverse sets for multiple queries.

respective grid cell c_l that corresponds to the multi-dimensional coordinates of x_j. A cell c that contains points from two or more result sets is considered to be in the "overlapping region" for those specific sets (Figure 4.3).

At this point, it is important to classify the types of overlap detected by DivM into the following:

1. **Point-based overlap:** Under point-based overlap, two data items are considered similar if they have the same exact values across all the grid dimensions.

2. **Cell-based overlap:** Under cell-based overlap, two data items are considered similar if they are mapped to the same grid cell.

Clearly, point-based overlap detects the *exact* natural overlap among search results, in which different search results might contain identical items (i.e., points). Meanwhile, cell-based overlap introduces *approximate* overlaps, in which different items from the same search result or different search results are considered identical.

Based on the above classification, we can broadly depict our proposed DivM scheme into: 1) DivM-Point, and 2) DivM-Cell. In particular, DivM-Point exploits the exact notion of overlap (i.e., point-based overlap), whereas DivM-Cell extends and generalizes DivM-Point via exploiting the more relaxed notion of cell-based overlap. As expected, each of those two version provides its own trade-off between processing cost vs. quality of diversification.

DivM-Point

DivM-Point, as outlined in Algorithm 6, recognizes and leverage the natural overlap in data that is expected to occur during the concurrent diversification of multiple results (i.e., point-based overlap). Hence, after finishing the two pre-processing steps listed in Section 4.2.2, the common items across the different search results are detected (Algorithm 6: line 1). In particular, each cell is further processed to detect identical data items without introducing any approximation. In that sense, the grid structure acts merely as a hashing scheme to reduce the costs of detecting

identical data items. That is, only the data items that are mapped to the same cell are further compared to detect if they are identical instead of comparing all the data items in all the results sets. DivM-Point then proceeds as outlined in Algorithm 6. In particular, all the search results are processed concurrently and in each iteration, one item of X_i is selected for inclusion in S_i, $1 \leq i \leq N$. DivM-Point exploits the overlap in search results as well as in diverse sets as explained in Section 4.2.2 and as outlined in Algorithm 6: lines 7–11.

Referring back to Example 3, the processing cost $C(S_1, S_2)$ incurred by DivM-Point in detecting the diverse set for X_1 and X_2 in that example can be expressed as:

$$C(S_1, S_2) = C_{\text{Naïve-Greedy}}(S_1, S_2) - O(k|S_1 \cap S_2||X_1 \cap X_2|)$$

Example 3 can be easily extended to the general case of N result sets (Figure 4.4), in which the cost $C(\mathbb{S})$ for processing n result sets (X_1,X_2, .., X_N) is expressed as:

$$C(\mathbb{S}) = C_{\text{Naïve-Greedy}}(\mathbb{S}) - O(k(\sum_{i=2}^{N} \binom{N}{i} |S'||X'|))$$

where S' is the intersection of two diverse subsets, and X' is intersection of two result sets.

Compared to Naïve Greedy, DivM-Point produces exactly the same set of diverse sets (S_1, S_2, .., S_N). During each iteration, however, DivM-Point clearly requires less operations for the computation and comparison of distance functions. Note that in addition to the processing costs required to locate the diverse sets, DivM-Point incurs additional costs for detecting the identical items in each cell. Clearly, that cost is dependent on different factors including grid resolution and data distribution. However, the reductions provided by DivM-Point during locating the diverse sets clearly outweighs that overhead (please see Section 4.4).

While DivM-Point is based on the general DivM approach, it introduces no approximations. Next, we introduce DivM-Cell, which introduces further flexibility in tuning the trade-off between efficiency and effectiveness.

DivM-Cell

Clearly the amount of reductions in processing time provided by DivM-Point is dependent on two orthogonal factors: (i) the amount of overlap among the diverse sets, and (ii) the amount of overlap among the result sets. For instance, consider again Example 3. In that example, for a sufficiently large $|X_1 \cap X_2|$, DivM-Point will provide significant gains over Naïve Greedy only if $|S_1 \cap S_2|$ is also large enough. This amount of gain, however, is expected to decrease with the decrease in $|S_1 \cap S_2|$ until it eventually disappears as $|S_1 \cap S_2|$ approaches zero, despite of

a large $|X_1 \cap X_2|$. Hence, an overlap in the result sets X_i cannot necessarily guarantee any performance gains in its own and is dependent on the amount of overlap in the diverse sets S_i. Here, we propose DivM-Cell, which overcomes that limitation.

DivM-Cell exploits the more relaxed cell-based notion of overlap to reduce the computational costs of diversification by orders of magnitude while achieving high levels of quality in diversification.

To illustrate that point, consider again two user queries Q_1, and Q_2 and their respective result sets X_1, and X_2 and diverse sets S_1, and S_2. Hence, it is possible to have two subsets S'_1 and S'_2 that are selected for inclusion in S_1 and S_2, respectively, such that each item in S'_1 is close to some item in S'_2, but not identical. In that case, there will be no natural overlap between S_1 and S_2 for DivM-Point to exploit (i.e., $|S_1 \cap S_2| = 0$). However, if the individual results (i.e., items) in S'_1 and S'_2 that are very similar to each other were considered the same, then this overlap between the diverse subsets would increase and we could exploit this fact to reduce the total computational cost. That is precisely the intuition underlying DivM-Cell algorithm (Algorithm 7), which involves the following steps:

1. **Selecting grid representatives.** DivM exploits the cell-based notion of overlap by selecting a *representative* point for each cell. This representative may be an actual point in the cell or a "virtual" point placed at the geometric center of the cell. Here, we choose as representatives the points that are closest to the geometric center of their respective cell. The representative of cell c is denoted as $Rep(c)$.

2. **Approximating result sets.** Given the grid representatives, each data point x_j in a cell c_l is approximated by $Rep(c_l)$. In turn, each set of results X_i is approximated by a set of representatives X_i^R. It is expected for $|X_i^R|$ to be much smaller than X_i, since many of the original points could be mapped to the same cell. This approximation is also expected to increase the overlap among the various result sets $\mathbb{S}$. That is, different items from different search results that are sufficiently similar to each other are considered identical. In either case, the impact of approximation depends on many factors including: grid resolution, result dimensionality, and the distribution of results over the data space.

3. **Locating diverse sets.** In DivM, the diverse set S_i for each X_i is evaluated by invoking the DivM-Point algorithm (see Section 4.2.2). To leverage the introduced approximations, however, DivM-Point is invoked to operate on sets of representatives (i.e., X_i^R generated in step 1) rather than the original sets of results (i.e., X_i). Similarly, in each iteration, one representative is selected to be added to the approximated diverse result set S_i^R,

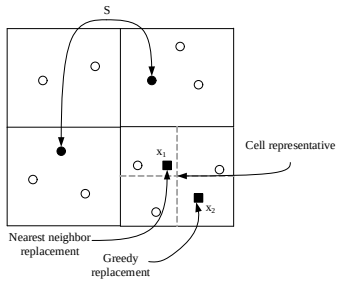


Figure 4.5: Alternative replacement options for DivM

$1 \leq i \leq N$. Hence, each result s_i^R in S_i^R is basically a representative of some cell c_l (i.e., $Rep(c_l)$).

In comparison to DivM-Point, DivM-Cell provides the following advantages:

- It represents each search result set X_i by a smaller approximated set X_i^R, which results in less processing and significant cost reductions.

- It creates overlap in search results (i.e., X) by considering points that belong to the same cell as similar, which also translates into higher degrees of overlap between diverse sets (i.e., S).

Although DivM-Cell uses only representatives of search results to evaluate diverse subsets, the diversity of S_i^R provided by DivM-Cell is comparable to S_i computed by DivM-Point as proved next.

Algorithm 7 DivM-Cell.

Input: Result sets $X_1, \ldots, X_n$, an integer k, a resolution γ.

Output: Diverse sets $S_1, \ldots, S_N$ with the k most diverse results of $X_1, \ldots, X_N$ respectively.

1: $\mathcal{G} \leftarrow$ Construct grid with resolution γ
2: map $X_1 \cup \ldots \cup X_n$ to $\mathcal{G}$
3: select a representative $Rep(c)$ for each cell c in $\mathcal{G}$
4: **for** $i = 1$ **to** n **do**
5: $\quad X_i^R = \{Rep(c_j) : x_j \in X_i$ with x_j mapped to $c_j\}$
6: **end for**
7: $S_1, \ldots, S_n \leftarrow$ DivM-Point($(X_1^R, \ldots, X_n^R)$, k)
8: refine $S_1, \ldots, S_N$
9: **return** $S_1, \ldots, S_N$

4.2.3 DivM Refinement

As such, the value of the grid resolution γ acts as a "knob" for tuning the amount of approximation introduced by DivM. Moreover, as described so far, a result s_i^R in the approximate diverse set S_i^R is basically a representative of some result that should be included in the actual diverse set S_i but not the exact one. In fact, S_i^R might also contain results that belong to other sets X_j, such that $j \neq i$. For those reason, DivM employs an additional *refinement step* to replace those representatives with more accurate results in order to transform S_i^R into S_i. In the following, we discuss several alternatives for performing such refinement in terms of: (i) method of refinement (i.e., how to refine), and (ii) timing of refinement (i.e., when to refine).

Regarding the method of refinement, DivM first maps s_i^R to its corresponding cell c_l. This is a straightforward step since $s_i^R = Rep(c_l)$. Then, the goal is to replace s_i^R with a point s_i in c_l that is as close as possible to the one that would have been generated without introducing any approximation. Towards this, we have considered the following two alternatives:

1. *Nearest Neighbor Replacement.* s_i^R is replaced with s_i, such that: (i) $s_i \in X_i$, and (ii) s_i is the nearest neighbor to s_i^R (i.e., $d(s_i, s_i^R)$ is minimum).

2. *Greedy Replacement.* s_i^R is replaced with s_i, such that: (i) $s_i \in X_i$, and (ii) s_i has maximum distance from the current instance of S_i^R (i.e., $d(s_i, S_i^R)$ is maximum).

In terms of efficiency, Nearest Neighbor Replacement requires less number of distance computations and comparisons than a Greedy Replacement. For example, if there are m points that belong to X_i in c_l, Nearest Neighbor replacement will perform only m distance computations to find a point closest to $Rep(c_l)$, whereas Greedy replacement will perform $m|S_i^R|$ distance computations. On the other hand, the overall diversity provided by Greedy replacement is expected to be higher than or equal to that provided by the alternative Nearest Neighbor replacement. This is because Greedy replacement tries to select the best possible point in c_l to replace $Rep(c_l)$ as it employs the same logic of the Greedy algorithm for diversification but at the local-level of a cell. For instance, as shown in Figure 4.5, x_2 selected by Greedy replacement has bigger distance from set S than x_1 selected by Nearest Neighbor replacement. In terms of the timing of refinement, we have also considered the following two alternatives:

1. *Eager Refinement.* The refinement step is applied at the end of each iteration. That is, as soon as s_i^R (or equivalently, $Rep(c_l)$) is selected to be added to S_i^R, it is immediately replaced by another point from c_l according to one of the refinement methods above.

Table 4.1: Evaluation Parameters.

Parameter	Description	Range	Default
m	Data size	5k–1500k	40k
D	Data dimensionality	2–10	2
N	Number of queries	2–1000	20
k	Diverse subset size	10–100	100
γ	Grid resolution	0–1	0.025
O	Overlap of queries	0%–100%	-

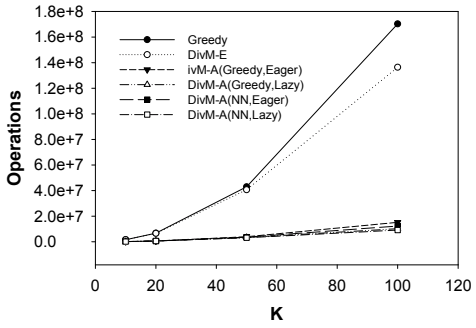


Figure 4.6: DivM vs Greedy when varying k (Cost)

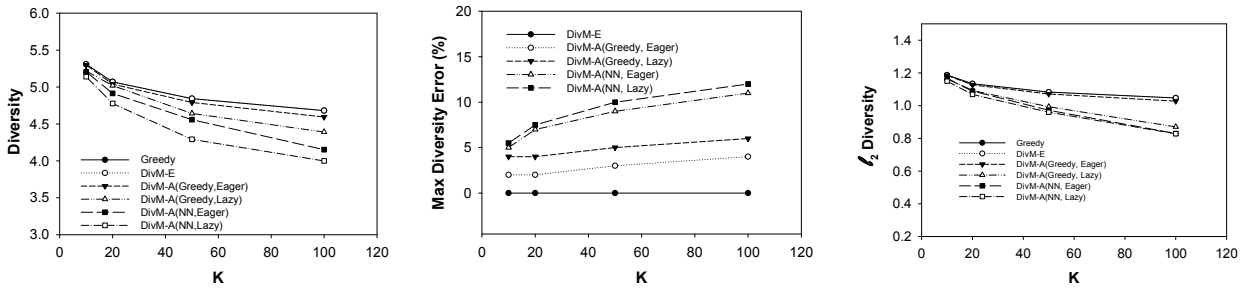
2. *Lazy Refinement.* The refinement step is applied once at the end of the algorithm. That is, once all the k most diverse results are added to S_i^R, it is scanned again and each s_i^R (or equivalently, $Rep(c_l)$) is replaced by another point from c_l according to one of the refinement methods above.

Clearly, eager refinement leverages approximation to only introduce overlaps between the result sets, but not the diverse sets, for which it relies only on natural overlaps for cost reductions. Meanwhile, lazy refinement allows virtual overlap in both diverse sets and result sets, which is expected to result in significant cost reductions.

DivM-Cell controls the trade-off between approximation and efficiency via tuning the grid resolution as well as configuring the refinement step described above. In the next section, we evaluate the impact of those parameters and others on the performance provided by DivM.

4.3 Experimental Testbed

We perform a number of experiments to evaluate the efficiency and the effectiveness of our DivM scheme. In particular, we compared the performance of our DivM algorithms, presented



(a) Diversity (b) Max Diversity Error (c) l_2 Diversity

FIGURE 4.7: DivM vs Greedy when varying k (Diversity).

in Section 4.2, against the baseline approach of using the Naïve Greedy algorithm for multiple search result diversification. Table 4.1 reports the various parameters used in our experiments throughout this section.

Schemes: We evaluate the following schemes:

- Greedy: Apply the Greedy heuristic independently on each result set to select the respective diverse subset (Naïve Greedy in Section 4.2.1).
- DivM-E: Process common search results only once for all queries and combine results using partial aggregation techniques (DivM-Exact in Section 4.2.2).
- DivM-A: Use spatial proximity between data via a grid mapping to increase the overlap among the various result sets (DivM-Approximate in Section 4.2.2). For which, we evaluate the following alternatives for refinements: DivM-A(NN, Lazy), DivM-A(Greedy, Lazy), DivM-A(NN, Eager), DivM-A(Greedy, Eager).

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

- Cost $C(\mathbb{S})$, measured as the sum of operations performed to locate set $\mathbb{S}$ of N subsets $\{S_1, S_2, \ldots, S_N\}$, where each operation represent one distance computation and data comparison task.
- Diversity $D(\mathbb{S})$, measured as $\sum_{S_i \in \mathbb{S}} f(S_i, d)$, across the set of diversified subsets $\mathbb{S}$. Moreover, in order to get a better understanding of the performance of our scheme and its fairness, we also report the l_2 norm of diversity given as $\sqrt{\sum_{S_i \in \mathbb{S}} (f(S_i, d))^2}$ as well as the perceived maximum error (i.e., inaccuracy) in diversity.

Data sets: We use both synthetic and real data sets. Our synthetic data sets consist of points in the D-dimensional Euclidean space, where D is a simulation parameter. Points are

either uniformly distributed ("Uniform") or form clusters around a random number of points ("Clustered"). Our real data set "Cities" contains 2-dimensional points representing the locations of 5922 towns (previously used in [31]). For all the data sets, dimensions are normalized in [0–1].

Queries: We generate a random set of range queries. For each experiment, the number of queries n is in the range [2–1000]. Further, we introduce a simulation parameter O to tune the amount of overlap between search results of any two range queries, where O varies between 0% to 100%. Each query is also associated with the size of diverse set k, which takes values in the range [10–100].

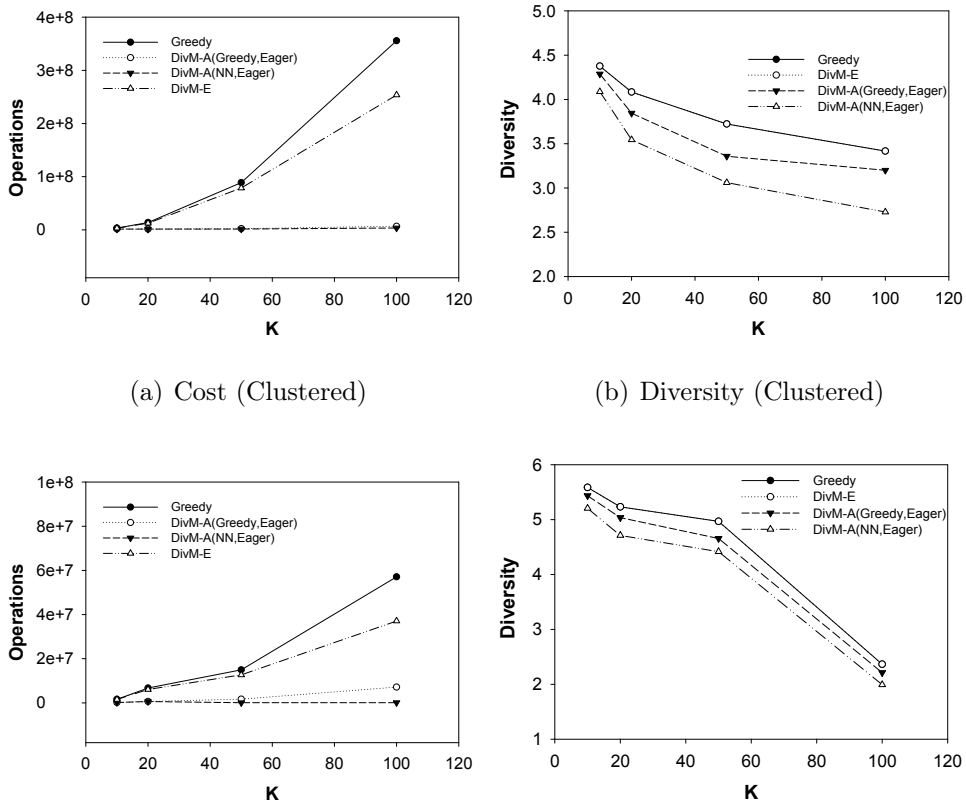
4.4 Experimental Evaluation

In the following experimental results, we report average values over ten runs with different sets of random range queries.

4.4.1 Impact of 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 4.6 shows the average number of operations performed by Greedy and our DivM algorithms for different values of k. We see that, as the value of k increases, the number of performed operations increases for all schemes. DivM-E, however, is performing up to 20% less operations when compared to Greedy. The cost savings are larger for larger values of k. The reason for this is that, for larger values of k, even a small percentage of overlap between result sets will provide more shared diverse results. However the cost savings for DivM-A increase along k, by more than 94% on average for all values of k.

Figure 4.7(a) shows that the overall diversity decreases with k for all schemes. The diversity achieved by DivM-E is equal to that achieved by Greedy. This is due to the fact that no approximation techniques are used. To the contrary, DivM-A operates on the representatives of the data, which essentially introduces virtual overlap among search results. Naturally, this leads to DivM-A achieving less diverse solutions than DivM-E. This loss in diversity, however, is marginal when compared to the gains achieved in efficiency. For instance, while the loss in diversity offered by DivM-A varies between 2% to 14%, the benefits it provides in terms of cost savings is more than 95%.



(a) Cost (Clustered)

(b) Diversity (Clustered)

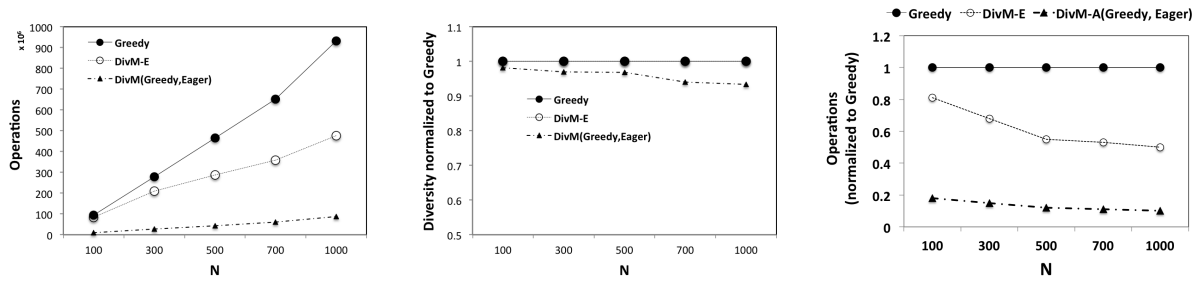
(c) Cost (Cities)

(d) Diversity (Cities)

FIGURE 4.8: DivM vs Greedy when varying k (Clustered and Cities data sets).

In order to capture the trade-off between the worst case diversity (Figure 4.7(b)) and the average case one (Figure 4.7(a)), the diversity achieved by each scheme in form of l_2 norm is shown in Figure 4.7(c). All the three metrics show that the diversity of DivM-A is comparable to DivM-E. Further, Figure 4.6 shows that all DivM-A variants perform orders of magnitude less operations than both Greedy and DivM-E. Among those variants, DivM-A(Greedy, Eager) provides better diversity (Figure 4.7(a)) but clearly at the expense of higher cost. For clarity of presentation, in the rest of this section, we only present the results of DivM(Greedy,Eager), unless otherwise mentioned.

Figure 4.8 reports the performance on clustered and cities data sets. As expected, when mapping clustered data to a grid, most of the search results fall in few cells. This leads to an interesting case where for some queries the number of representatives selected by DivM-A might be less than k, which renders the lazy refinement method inapplicable. Therefore, we report the results of eager versions of DivM-A. Figure 4.8(a) shows that DivM-E performs up to 20% less operations as compared to Greedy. DivM-A achieves up to 94% reductions in number of operations. Figure 4.8(b) shows that for clustered data set, DivM(Greedy,Eager) achieves diversity within 6% of that of DivM-E. However, the diversity of DivM-A(NN,Eager) declines



(a) Cost vs. number of queries. (b) Diversity vs. Number of queries. (c) Cost Savings.

Figure 4.9: Impact of Number of Queries.

when applied to that same data set. This is due to the fact that when the number of representatives is less than k, applying the Nearest Neighbour technique to select new points from already selected cells will have adverse effect on the diversity of the final subset. Figures 4.8(c) and 4.8(d) show similar performance for the cities data set in terms of cost and diversity, respectively.

4.4.2 Impact of Number of Queries

Figure 4.9(a) shows that the number of performed operations increases for all the algorithms with increasing number of queries. The cost savings achieved by DivM-E increase as N approaches 1000, and reaches upto 50%. However, as shown in Figure 4.9(c), the rate of increase in cost savings tends to slow down for higher values of N. This is due to the higher overhead of detecting overlapping results among large number of queries For instance, the difference in percentage of cost savings between 300 and 500 queries is around 7, whereas the difference in percentage of cost savings between 500 and 700 queries is only 3. Meanwhile, DivM-A performs consistently and achieves significant cost savings. As DivM-A processes only small number of representative results, the overhead of detecting overlapping results is compensated by the massive reductions in the data diversification costs. For instance, as Figure 4.9 shows, the cost reductions provided by DivM-A are up to 90% for $N = 1000$. Figure 4.9(b) shows that diversity of N sets evaluated by DivM-A decreases by at most 7% with increasing the number of queries. This decrease is due to the presence of more overlapping regions that result in higher number of approximated points in diverse subsets, which is evaluated next.

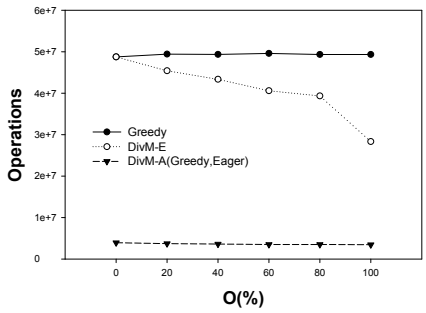


FIGURE 4.10: Impact of Query Overlap on operations.

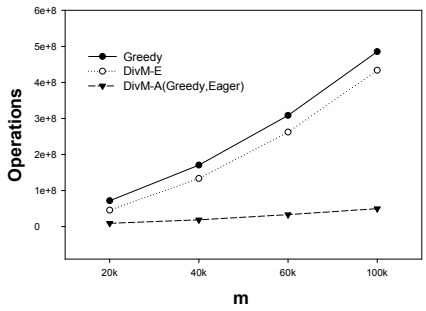


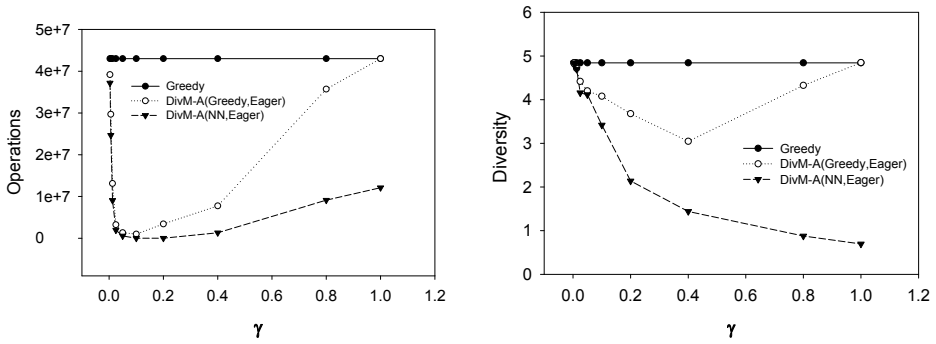
FIGURE 4.11: Impact of Data Size on Operations.

4.4.3 Impact of Query Overlap

To study the impact of the overlap between queries, we first generate one query Q_1 for our "Uniform" dataset. Then, we generate an identical second query Q_2 and we "slide" Q_2 over Q_1 to control the overlapping area between the result sets of the queries. Figure 4.10 shows that the number of operations performed by Greedy are not affected by the change in data overlap. However, the cost of DivM-E and DivM-A decreases as the amount of overlap increases, since in that case, there is more data overlap that can be exploited by both schemes. The maximum performance gain is achieved when the two queries have identical result sets, i.e., when their overlap is 100%. In that case, the cost savings of DivM-E are around 50% and that of DivM-A around 99%.

4.4.4 Impact of Data Size

Next, we perform an experiment to see how our algorithms scale when the data size m is increased. Figure 4.11 shows that DivM-A scales well since it performs distance computations among the representatives of the various cells as opposed to the whole data set. Greedy and DivM-E scale linearly along m. Although DivM-E performs less operations than Greedy, the ratio in performance of DivM-E to Greedy is increasing. This is due to the fact that in a highly dense data space, it is highly unlikely that natural overlap occurs in diverse subsets.



(a) Cost vs grid resolution. (b) Diversity vs grid resolution.

Figure 4.12: Impact of Grid Resolution.

Meanwhile, the cost savings achieved by DivM-A increase from 88% to 92% as the number of representative points selected from the Grid depends upon the Grid resolution and not the data size. Hence, as more data points in each Grid cell are represented by one single point the relative cost savings increase.

4.4.5 Impact of Grid Resolution

Figure 4.12(a) shows the variations in number of performed operations along the employed grid resolution γ. At the two extreme cases, where $\gamma \simeq 0$ (i.e., each cell contains a single result or many identical results) or $\gamma = 1$ (i.e., there is only one cell in the grid), DivM-A is reduced to DivM-E and Greedy, respectively. In-between these two extremes, as the resolution increases, the number of operations decreases until a local minimum is reached and later increases again. This is because with increasing γ, the number of cells decreases. Hence, DivM-A algorithm evaluates fewer representatives. However when the number of representatives become less than k, DivM-A selects multiple items from the same grid cell. The cost of this process depends on the density of the cell. With higher values of γ we have highly populated grid cells, therefore the number of operations performed by DivM-A starts increasing. Among the DivM-A variants, DivM(Eager, Greedy) is the most expensive since its refinement process is more involved, especially for higher resolutions where each cell contains more results. However, DivM(Eager, Greedy) is the most effective variant in terms of diversity (Figure 4.12(b)).

4.4.6 Impact of Number of Dimensions

Finally, we perform an experiment to see how the dimensionality D of the data affect the performance of our algorithms. Figure 4.13 shows the corresponding results for one of our datasets "Uniform". DivM-E remains generally unaffected when D is increased. DivM-A,

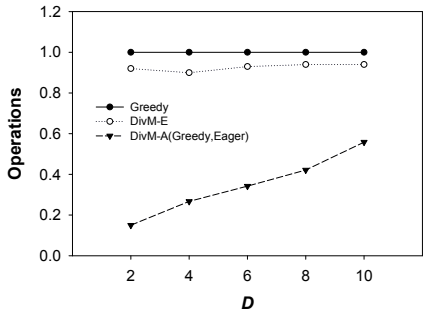


FIGURE 4.13: Impact of Number of Dimensions.

however, performs a larger number of operations. Clearly, this is a side-effect of the curse of dimensionality. That is, as D becomes larger, the data space becomes sparser and less results are mapped into a single cell. Therefore, as the number of dimensions increase, the number of representatives also increases and, thus, the computational cost of DivM-A is increased as well.

4.5 Summary

In this chapter, we focused on the NP-hard problem of diversifying the results of multiple queries and proposed DoS, an efficient scheme to locate approximate solutions. As compared to the existing techniques for multiple search result diversification, our proposed algorithm not only leverages the natural overlap in the search results but also increases this overlap by mapping the data space to a multi-dimensional grid. Our algorithm provides solutions of quality comparable to that of sequential methods, while significantly reducing processing costs as shown in our experimental evaluation.

Chapter 5

Sequential Diversification of Multiple Search Results

During Data Exploration, the user interaction with the database takes the form of an *exploratory session*, or session for short [15]. A session typically involves a lengthy sequence of related queries to retrieve interesting data objects. This requires processing of numerous queries that potentially return large number of results. Diversification of those query results adds additional computational cost to the exploration process. This highlights the need for efficient schemes for the diversification of multiple query results within a user session. In Chapter 4, we addressed the similar problem of diversifying the results of multiple concurrent queries. Our proposed *DivM* scheme utilizes the overlaps between query results to eliminate redundant computations. Although, sequential queries in a user session are also likely to access overlapping portions of data, those overlaps are not known a prior. Hence, in this chapter we propose an efficient scheme for sequential diversification in data exploration platforms

Our proposed scheme, called *AdOr* relies on two main interrelated components, namely: 1) an adaptive model-based diversification method, and 2) an order-based caching scheme. In particular, *AdOr* employs an adaptive model based diversification method to estimate the diversity of a diverse subset and hence selects diverse results without scanning all the query results. In order to further expedite the diversification process, *AdOr* employs an order-based caching scheme to leverage the overlap between sequence of data exploration queries. Specifically, aim of *AdOr* scheme is to balance the trade off between the efficiency of diversification (i.e., processing cost) and its effectiveness (i.e., accuracy of diversification) during data exploration.

In a nutshell the contributions of this work are as follows:

- We formulate the problem of Sequential Diversification for data exploration, together

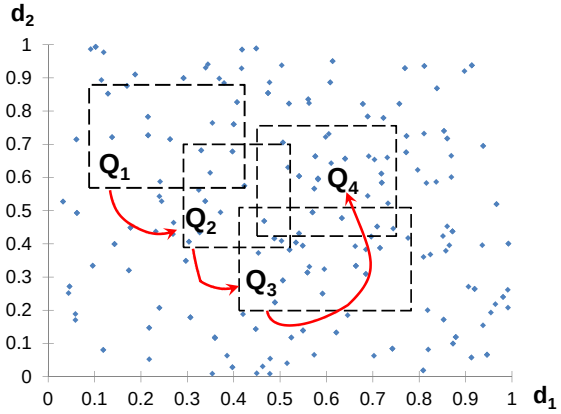


FIGURE 5.1: Series of exploratory queries generated in a user session

with metrics that captures both the efficiency and effectiveness of diversification.

- We propose the novel *AdOr* scheme, which utilizes an adaptive model-based approach, that is particularly suitable for the efficient and effective diversification in Data Exploration platforms.

- We propose an ordered-based caching scheme that leverage the overlap between query results within an exploratory session towards efficient diversification.

- 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 *AdOr*.

Rest of the chapter is organized as follows. We define the search result diversification for session based data exploration in Section 5.1. Next we introduce our *AdOr* scheme in section 5.2. The evaluation testbed and results are reported in Section 5.3 and Section 5.4 respectively. We compare our proposed approach to current approaches for multiple query diversification in Section 5.5 and we summarize the chapter in Section 5.6.

5.1 Sequential Diversification Model

In this section, we extend the notion of result diversification to the problem of sequential diversification during data exploration. In Table 5.1, we summarize all the symbols and notations used in the rest of the chapter.

Table 5.1: Table of Symbols.

Symbol	Description
N	Total number of queries in a user session
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
x_{max}	Result in X_c with maximum distance from S_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
γ	Model Threshold

5.1.1 Sequential Diversification

Scientists and analysts typically explore a relational database through a sequence of exploratory queries to discover interesting portions of data. 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. In order to guide the user throughout the data space, a data exploration platform is expected to manipulate the query result to help user understand the underlying data and further explore related queries.

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 [17]. This leads to an exploratory session, in which a user executes numerous selection queries iteratively using different predicates [28]. For example, consider a user searching for some interesting results in a two dimensional data space as shown in Figure 5.1. The user will first pose query Q_1 and after reviewing the results she might reformulate it into another query Q_2 and so on. The sequence of queries illustrated in Figure 5.1 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`

$0.430 < dim_2 < 0.750$.

Since, different queries within an exploratory session typically explore the data space in a close vicinity to each other, it is very likely for queries within a session to have overlapping results as shown in Figure 5.1. Hence, the diversificaition of multiple related queries within a user session can be viewed as an instance of *continuous diversification* problem in which a diverse subset needs to be computed for each sliding window over dynamic data stream. Each exploratory query can also be perceived as a sliding window. However, in dynamic data stream a sliding window is moving along the time dimension, whereas an exploratory query is moving in space over static data. The result set of each query is generated from the new subspace of data that it covers and a corresponding diverse subset of that result set needs to be computed. Hence, here we extend the definition of result diversification from chapter 2 to the diversification of multiple sequential queries as defined below:

Definition 12. *Let $\mathbb{M}$ represent a set of multiple queries within an exploratory session. That is, $\mathbb{M} = \{Q_1, Q_2, \ldots, Q_N\}$. The sequential diversification problem is the following: For each query Q_i in $\mathbb{M}$ with a result set X_i, compute a set S_i^*, such that:*

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

where

$$f(S_i, d) = \min_{\substack{x_i, x_j \in S \\ x_i \neq x_j}} d(x_i, x_j)$$

Note that, in this work we focus on the MaxMin diversity measure to define $f(S_i, d)$, as it adopts a more uniform view to represent all the results in X_i.

The challenge of computing diverse subsets for each query Q_i in $\mathbb{M}$ is that those diverse subsets can not be computed incrementally. For instance, let X_i and X_{i+1} be result sets of two overlapping queries in $\mathbb{M}$. Then diverse subset S_{i+1} can not be computed by updating the diverse subset S_i as the two diverse subsets may be completely different despite substantial overlap in the result sets X_i and X_{i+1}. Hence, each diverse subset S_i needs to be computed from scratch. Therefore, the computational cost of diversifying N queries in a session increases linearly with the increase in N. Hence, in this work we address the problem of designing efficient diversification methods that are scalable to N i.e., length of a user session.

5.1.2 Problem Definition

To capture the performance of solutions for sequential diversification during a data exploration session, we define the following metrics:

Definition 13. *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 a session of N queries is: $\frac{1}{N}\sum_{i=1}^{N} C(S_i)$.*

Definition 14. *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 a session of N queries is: $\frac{1}{N}\sum_{i=1}^{N} D(S_i)$.*

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 order to achieve this goal our proposed *AdOr* scheme utilizes an adaptive model to reduce the cost of diversification for each query in the exploratory session. *AdOr* further exploits the fact that the results of those queries are expected to exhibit some degree of overlap. Thus, *AdOr* leverage that overlap between query results by employing an order-based cache to store diverse results of previous queries. In the next section, we present our scheme and discuss its impact on both the efficiency and effectiveness of diversification.

5.2 AdOr Scheme for Sequential Diversification

In this section, we present our *AdoR* scheme for the efficient diversification of query results in data exploration platforms. The main idea underlying *AdoR* is to leverage an adaptive regression model to estimate the diversity of a subset and utilize the overlap in a sequence of queries (such as the one shown in Figure 5.1) in order to minimize the computational cost incurred in diversifying each query result set. Towards that goal, AdOr exploits two novel techniques, namely: 1) Regression model based diversification, and 2) Cache based diversification. Before going through the details of each of those techniques, we first present our baseline solution for the diversification of multiple sequential queries, namely: Greedy Construction.

5.2.1 Greedy Construction

In this baseline solution, the *Greedy Construction* algorithm (or *Greedy* for short), is directly applied for the diversification of results. In particular, Greedy (as presented in Algorithm 1) evaluates the diverse subset S of each query iteratively by computing set distances of all the results in a query result set X. For instance, in order to select a new result in partially computed

diverse subset S_i, Greedy computes the set distance of each result in X from results already selected in S as:

$$setDist(x, S) = \min_{\substack{x \in X \\ x_j \in S}} d(x, x_j)$$

The result x_{max} having the maximum set distance is added to S. Thus, in k iterations Greedy selects k results to be added to diverse subset S. Also, Greedy processes each query result individually without taking into consideration the overlapping in results between different queries.

Recall from Section 2.2.2, the computational complexity of diversifying a single query using Greedy algorithm is $O(k^2|X_i|)$, where $|X_i|$ is the size of the query result and k is the size of the diverse subset. Clearly, applying Greedy algorithm independently to each exploration query, results in a computational complexity that increases linearly with the increase in the number of total queries in a user session (i.e., N). That is, the complexity of Greedy algorithm for N queries is simply computed as: $O(k^2|X_1|) + \ldots + O(k^2|X_N)|)$.

5.2.2 Regression Model for Diversity

As mentioned above when diversifying results of the current query Q_c, in each iteration i of the Greedy algorithm, we have a partially computed diverse subset S_c^{i-1} of size $i-1$ such that $i - 1 \leq k$. The choice of next optimal result x_{max}^i is obviously made after examining all the candidate results in X_c. In this work, we take an alternate approach for identifying x_{max}^i. Instead of examining all the results in X_c, we aim 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 a result x_p that provides diversity value comparable to the estimated diversity value is found, then there is no need to search the query result set X_c any further. Thus if x_p is the p^{th} result in X_c then for remaining $|X_c| - p$ results, set distance computations are not performed.

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., [26, 79]). 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 $f(S, d)$.

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

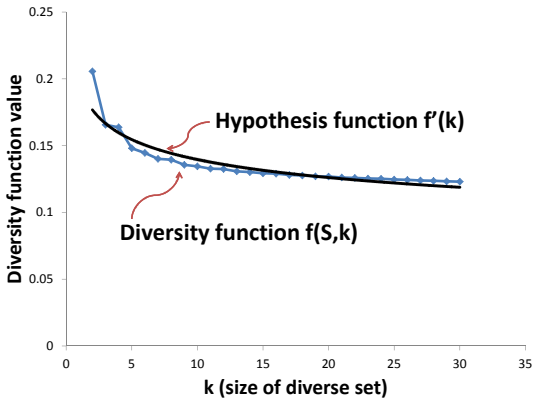


Figure 5.2: Diversity Curve.

a particular functional form ([68]). 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 ([68]).

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, we plot a *diversity curve* as shown in Figure 5.2 for a sample query result generated over a uniform data set (for more details on the different datasets we have experimented with, please see Section 5.3). The curve is generated by plotting the diverse subset size k on the x-axis and the diversity function value for the corresponding diverse subset on the y-axis.

As Figure 5.2 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 [12]. Such trend allows us to use a simple model like power series to model the diversity curve (Figure 5.2).

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.

As mentioned above, the regression model relies on observational data to evaluate the hypothesis function parameters. However, in case of query diversification, those observational values are only available once the query result has been diversified. Hence, in order to learn the regression parameters we consider two possible approaches. First, we present a baseline static approach and then we generalize it to an adaptive approach.

Static Model Approach The static model approach relies on the observations made using a sample query over the database. These observations are then used to evaluate the model parameters a and b in the hypothesis function. Once those parameters are known they remain static across various user queries. For instance, to learn the regression model parameters, a global sample query Q_G is executed over the database D to retrieve sample result set X_G. The Greedy algorithm is then applied over X_G to select a diverse subset S_G of size k. In each iteration i, where $i \leq k$, the diversity function value $f(S_G^i, d)$ is evaluated and the pair $(i, f(S_G^i, d))$ is plotted. As Greedy finishes execution, we have $k-1$ observations in the form of $(i, f(S^i, d)$ pairs. These observations are used for evaluating values of parameters a and b. Since, those values are based on observations generated by a single gobal query Q_G, they depend upon: (1) the size of the data subspace accessed by Q_G, and (2) the distribution of data within that subspace. Thus, the global regression model built based on data retrieved using global query Q_G covering whole data space is 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.

Adaptive Model Approach As discussed above, the Static Model Approach relies on the observation data generated using a sample global query Q_G. The regression model built using query Q_G can be scaled to be used for other queries provided the data space is uniformly distributed. However, for clustered data, static model approach fails to adjust the model parameters for each user query effectively. This is due to the fact that in clustered data some of the queries are overly populated while others return only few results even if the size of the data space accessed is the same. Thus, even after scaling the regression model, the underlying

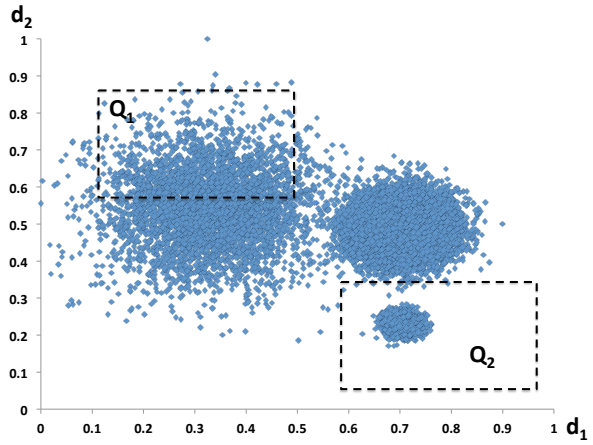


Figure 5.3: Interactive Diversification for Clustered Data.

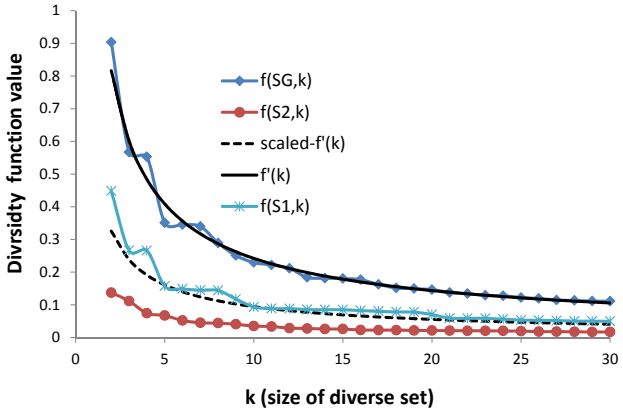


Figure 5.4: Diversity Curves.

data does not fit the model very well.

For instance, as shown in Figure 5.3, the size of data space covered by Q_1 is the same as the space covered by Q_2, however, the data distribution for both queries is very different. Although, the diversity values evaluated for both queries, for the same values of k, are expected to be quite different, scaling global regression model for both Q_1 and Q_2 will generate same values for parameters a and b. For example, Figure 5.4 shows the corresponding diversity curves for both Q_1 and Q_2. Clearly, the scaled regression model fits the diversity curve of Q_1 while fails to adjust to the diversity curve of Q_2.

To address the limitations of static model approach, we consider a more general approach based on adaptive regression model that learns model parameters individually for each user query and does not rely on any prior observation data. The main idea underlying the adaptive model approach is to utilize the observations from the current query being diversified. Instead of using an existing global model, for each individual query the regression model is built on the

go as we diversify the query. The adaptive model approach can be summarized in the following steps:

- Step 1: An initial diverse subset S_c of size two is generated using Greedy construction algorithm.
- Step 2: The diversity function value of S_c is used as first observation and initial values for model parameters a and b are evaluated.
- Step 3: Based on the hypothesis function, diversity function value is estimated as Div_e for the next iteration of Greedy algorithm.
- Step 4: New result with the maximum set distance from the already selected results in S_c is added to S_c. The actual diversity function value for newly updated set S_c is computed as Div_a and added to the obseravation data.
- Step 5: The difference in the actual diversity function value (Div_a)and the estimated diversity value predicted using hypotheis function(Div_e) is computed. The regression model is assumed to be stable if the following condition is true:

$$\gamma \geq \frac{Div_a - Div_e}{Div_e}$$

where γ is the threshold value defining the acceptable difference ratio between estimated diversity value and the actual diversity value. If the above condition is not met then the actual diversity value Div_a is added to the observation data. The new values of model parameters a and b are computed based on updated updated observations and Steps 3 to 5 are repeated.

5.2.3 Model based Greedy Algorithm

In this section, we present a model based Greedy algorithm called *mGreedy* for selecting diverse query results. We assume that the hypothesis function $f'(k) = ak^{-b}$ has been defined with parameters a and b using either static or adaptive approach. Note that, in case of adaptive approach some of the diverse results are already selected and *mGreedy* is applied to select the remaining diverse results. Let Q_c be the current query and X_c be the respective result set of Q_c. Like, Greedy algorithm, *mGreedy* also builds the diverse subset iteratively. In each iteration i, an estimated 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}, *mGreedy* uses a *deviation*

Algorithm 8 mGreedy

Input: Query Result set $X = (x_1, x_2..., x_m)$, Set of previous overlapping diverse results S_R, deviation threshold θ, a hypothesis function $f'(i)$, Partially computed diverse subset S, Diverse subset size k

Output: S with k diverse results

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

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, *mGreedy* selects a result $\bar{x}_{max}$, which has the deviation less than θ. As soon as such a result is identified the iteration is terminated. Hence, if $\bar{x}^i_{max}$ is the p^{th} result in X_c then this saves $|X_c| - p$ set distance calculations for the remaining candidate results in X_c. It is important to mention here that $\bar{x}^i_{max}$ is an approximation of x^i_{max}. Thus, instead of adding x^i_{max} to S_c^{i-1} the approximated result $\bar{x}^i_{max}$ is added to S_c^{i-1} . Therefore, $f(S_c^{i-1} \cup \{x^i_{max}\}, d) \geq$

$f(S_c^{i-1} \cup \{\bar{x}_{max}^i\}, d)$.

Applying *mGreedy* individually to each query reduces the cost of diversification, however, in the worst case scenario it might have to look at large number of results before it locates $\bar{x}_{max}^i$. In order to locate the promising results quickly we make use of the overlapping diverse results from the previous queries. In the next section we discuss how AdoR exploits caching to leverage the overlap between different queries and reduces the cost of diversification even further.

5.2.4 Cache based Sequential Diversification of Overlapping Queries

A data exploration session generally involves multiple related queries, overlapping between results of those queries is naturally expected to occur. 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 15. *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 16. *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 17. *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., $S_{O,c} \cap X_c$).*

Algorithm 9 AdoR

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 $f'(i)$

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

1: $prediction \leftarrow false$
2: $i \leftarrow 1$
3: $x \leftarrow$a random result in X_c
4: $S^i \leftarrow \{x\}$
5: $i \leftarrow i + 1$
6: **if** f'(i)= Null **then**
7: $x_{max} \leftarrow argmax_{x_{max} \in X_c} min_{x_j \in S^{i-1}} d(x_{max}, x_j))$
8: $S^i \leftarrow \{x_{max}\}$
9: $O \leftarrow O \cup \{(i, f(S^i, d))\}$
10: $f'(i) \leftarrow BuildModel(O)$
11: $i \leftarrow i + 1$
12: **while** NOT (StableModel) **do**
13: $estdiv \leftarrow f'(i)$
14: $x_{max} \leftarrow argmax_{x_{max} \in X_c} min_{x_j \in S^{i-1}} d(x_{max}, x_j))$
15: $S^i \leftarrow \{x_{max}\}$
16: **if** $(Ratio(estdiv, f(S^i, d)) \leq \gamma)$ **then**
17: $StableModel \leftarrow true$
18: **else**
19: $O \leftarrow O \cup \{(i, f(S^i, d))\}$
20: $f'(i) \leftarrow BuildModel(O)$
21: **end if**
22: $i \leftarrow i + 1$
23: **end while**
24: **end if**
25: $S^k \leftarrow S^i \cup mGreedy(X_c, S_{R,c}, S^i, \theta, f'(i), k - i)$
26: **return** S^k

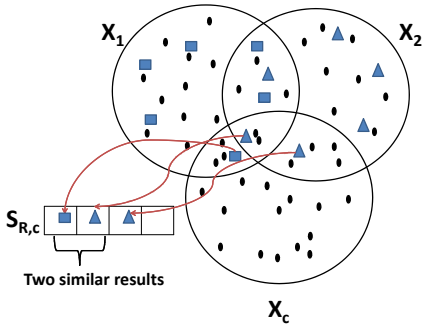


FIGURE 5.5: Reusable Set $S_{R,c}$ for current query result set X_c generated from diverse overlapping results of previous queries

Thus, one idea towards utilizing overlapping result set $S_{R,c}$ across multiple queries 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 5.5 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 5.5, 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.

Taking advantage of natural overlap occurring between various queries during an exploration session 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 AdoR scheme. Next we present in detail how the AdoR scheme achieves this goal.

Dividing the Search Space

Recall that in each iteration of the **m**Greedy algorithm, the first result that provides the diversity close to the estimated diversity value as predicted by the regression model is added to the diverse subset. Clearly, with random order of results, the optimal result may reside

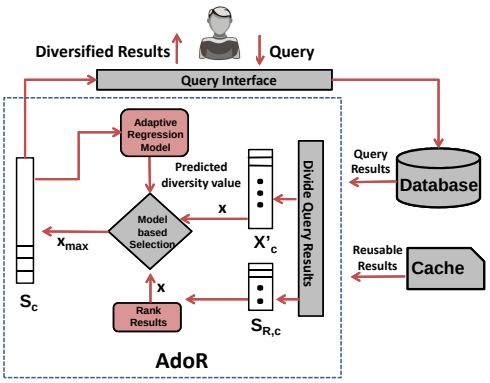


Figure 5.6: AdoR Architecture.

in the end of the result list requiring many set distance computations for results appearing before it. For large data sets where numerous results are returned in response to each query, and where near real time performance is expected, the goal of AdoR is to minimize the cost incurred in locating the result $\bar{x}^i_{max}$ in potentially large search space defined by the results in X_c. To minimize that cost, AdoR *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, $X'_c = X_c \setminus S_{R,c}$

Selection of Diverse Results Using Cache

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 $S_{R,c}$, AdoR 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 guided by the regression model so that to avoid compromising the quality of diversification.

As shown in Figure 5.6, to diversify a result set X_c, AdoR divides X_c into two subsets: the reusable diverse result set $S_{R,c}$ and the new query result set X'_c. For selecting that result $\bar{x}^i_{max}$, AdoR uses the following two alternative selection methods:

- **First Fit Diversification (AdoR-FF):** Search the set $S_{R,c}$ for the first result $\bar{x}^i_{max}$, which has the deviation value less than the deviation threshold θ. If such a result is

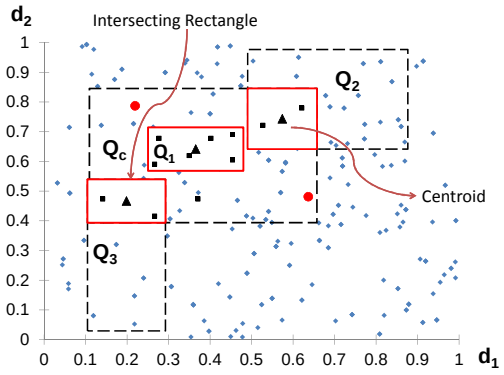


FIGURE 5.7: Ordering of overlapping cached diverse results.

found, it is added to set S_c^{i-1}, else X'_c is searched until $\bar{x}^i_{max}$ is located.

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

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, AdoR-FF picks the first result that provides comparable diversity to the one predicted by the regression model. Whereas while searching $S_{R,c}$, AdoR-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 AdoR-BF examines all the results in $S_{R,c}$. Thus, AdoR-BF tries to find diverse subsets with higher diversity at the expense of higher computational cost. The working of the AdoR scheme are presented in Algorithm 9. The Algorithm 9 shows the general adaptive model. However, to employ the static approach the input parameter $f'(i)$ can be initialized by a hypothesis function computed a prior using a static model.

Ordering of cached diverse results

Obviously, the performance of AdoR-FF depends a lot on the order in which the results are stored in $S_{R,c}$. If the results distant from the current partial diverse subset $S_c^i - 1$ appear earlier in the search list then the cost savings are higher. Keeping this goal in mind, we propose an ordering scheme that processes the promising overlapping diverse subsets first. In order to elaborate the ordering scheme consider the following example.

Let Q_c be the current query with result set X_c for which a diverse subset S_c needs to be generated. Also, let $\mathbb{Q}_{O,c}$ be set of three previous queries overlapping with Q_c containing Q_1, Q_2 and Q_3 as shown in Figure 5.7 . In a random iteration i of greedy heuristic the diverse results already selected in S_c^i are shown in red circle shapes. The overlapping diverse results from the previous queries are shown in square shapes. As clear from the Figure 5.7, the results in Q_3 are most distant from the results already in S_c^i. However, since the results are stored in the order in which the queries were generated, the results in Q_3 will be examined last.

Thus, in order to prioritize the order in which queries from $\mathbb{Q}_{O,c}$ are examined, we identify the intersecting rectangles between Q_c and each of the overlapping cached queries in $\mathbb{Q}_{O,c}$ as shown in solid boundaries in Figure 5.7. Next, we determine the centroid of each of these intersecting rectangles. Those centroids represent their respective query in $\mathbb{Q}_{O,c}$ and are shown as diamond shapes in Figure 5.7. The set distance between each query Q in $\mathbb{Q}_{O,c}$ from the diverse subset S_c^i is calculated as:

$$setDist(Q, S_c^i) = setDist(centroid_Q, S_c^i)$$

The queries with higher set distance from the diverse subset S_c^i have the higher potential of containing $\bar{x}_{max}^i$ as compared to the queries with smaller set distances from S_c^i. Hence, each query in $\mathbb{Q}_{O,c}$ is assigned a priority score as: $Score^i(Q) = setDist(Q, S_c^i)$. All the queries in $\mathbb{Q}_{O,c}$ are then processed in the decreasing order of their scores. It means the diverse results from the most distant overlapping query are processed first.

It should be noted that $Score^i(Q)$ is the priority of Q with respect to S_c^i in iteration i. As the diverse subset S_c evolves in subsequent iterations the score of each query in $\mathbb{Q}_{O,c}$ also changes. Hence, in each iteration the score of each overlapping query is re-evaluated by computing a set distance between the centroid of the query and the diverse subset S_c.

The additional cost of computing priority scores in terms of number of set distance computations is equal to the number of overlapping queries ($|\mathbb{Q}_{O,c}|$). Since, the number of overlapping queries is usually much small as compared to the number of reusable diverse results ($|\mathbb{Q}_{O,c}| \ll |S_{R,c}|$), overall cost savings in terms of number of distance computations can still be expected. However, with increasing number of queries in cache the number of queries overlapping with the Q_c can also increase. This will not only have an impact on the computational cost of priority scores but will also increase the storage cost. Hence, in order to keep the number of cached queries limited yet achieving the benefits of previously cached results we employ some effective cache management techniques as discussed next.

TABLE 5.2: Adaptive Model Schemes.

-	No Cache	Random Cache	Ordered Cache
First Fit	Adaptive-FF-NoCache	Adaptive-FF-RandomCache	Adaptive-FF-OrderedCache
Best Fit	Same as Greedy	Adaptive-BF-RandomCache	Same as Random Cache

5.2.5 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.
- The computational cost of evaluating scores for ordering overlapping queries may out weigh the benefits of ordering.

As a consequence of the observations above, a large cache size would reduce the amounts of savings provided by AdoR until it reaches zero. That is, the processing cost of AdoR becomes similar to that of Greedy. The obvious approach to address these challenges is to limit the reusable set $S_{R,c}$ size. That is, instead of using the entire reusable set $S_{R,c}$, a subset $\mathbb{S}'_{R,c}$ is used. Accordingly, as new queries are posed, some of the diverse subsets are evicted from cache to make room for new subsets. Hence, when the number of diverse subsets become greater than L, AdoR 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.

5.3 Experimental Testbed

We perform a number of experiments to evaluate the efficiency and the effectiveness of our AdoR scheme. In particular, we compare AdoR against two baseline approaches: Greedy and Static Model Approach. Table 6.3 summarizes the different parameters used in our experimental evaluation.

Table 5.3: Evaluation Setting.

Parameter	Range	Default
Number of Queries (N)	2–1000	100
Diverse Subset Size (k)	10–40	30
Number of Cached Queries (L)	20–100	20
Deviation Threshold (θ)	-	0.05
Model Threshold (γ)	0.01–0.05	0.02
Data Size (D)	20k–40k	20k
Data sets	Unif., Clust., SDSS	Unif.

Schemes: We evaluate the performance of the following schemes:

- **Greedy:** Applies the Greedy Construction heuristic independently on each query result set to select the respective diverse subset.

- **SGC:** Uses Stream Greedy Construction heuristic as presented in [30]. SGC relies on the overlap that occurs between the results of two consecutive queries (i.e., $X_i \cap X_{i-1}$). The diverse subset S_i is initialized with the r overlapping diverse results from S_{i-1}. Thus only $k - r$ remaining diverse results are computed using Greedy heuristic. The details of SGC scheme are given in section 5.5.

- **Static-No-Cache:** Uses Static Model approach to build the regression model based on sample observations generated by a global query. The regression model is then used to predict future values of diversity function for the selection of diverse subsets across various queries.

- **Adaptive-No-Cache:** Applies Adaptive Model approach to build a regression model for each individual query.

- **Adaptive-Random-Cache:** Extends Adaptive Model approach to use diverse results of previous overlapping queries stored in cache. The diverse results from cache are accessed in random order.

- **Adaptive-Ordered-Cache:** Employs a priority scheme to order the results of overlapping queries in cache. The diverse results in cache are accessed in decreasing order of their distance from already selected diverse results for the current query.

For all the schemes using cached overlapping diverse results, we further evaluate both best fit and first fit alternatives.(Section 5.2.2). All the variations of AdoR scheme are summarized in Table 5.2

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 ("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].

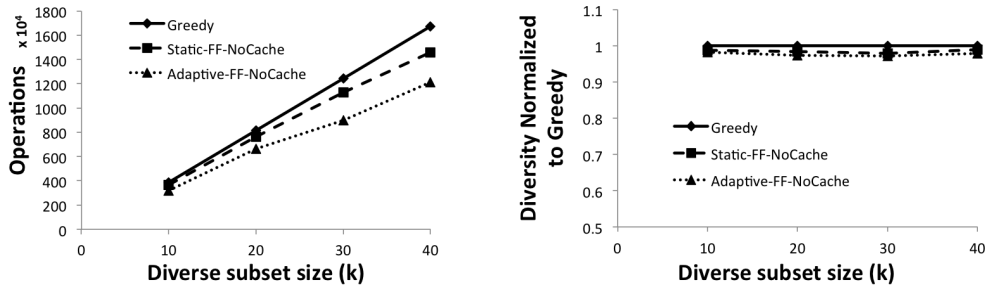
Queries: We simulate random user sessions with multiple range queries. For each experiment, the number of total queries N, across sessions is in the range [2–1000]. Each query is also associated with the size of diverse set k, which takes values in the range [10–40]. Cache is initialized with diverse results of 20 queries.

5.4 Experimental Evaluation

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

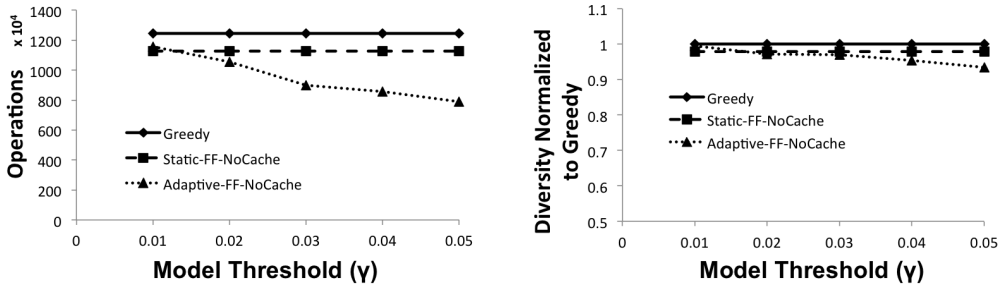
5.4.1 Impact of Adaptive Regression Model

In this experiment we compare the performance of Adaptive model approach against the Static model approach. In this experiment all the queries are generated over clustered data set. We compare the performance of both schemes without using any cached results to emphasize on the impact of regression model alone. Hence, under this experimental setting the performance of Best-Fit diversification approach is similar to Greedy. Therefore we compare the performance of only First-Fit alternatives by varying the following parameters.



(a) Cost (b) Diversity

Figure 5.8: Adaptive Model vs. Static Model



(a) Cost (b) Diversity

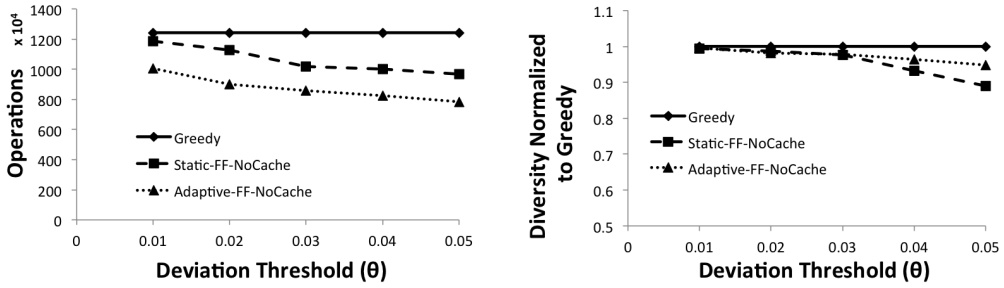
Figure 5.9: Impact of γ on Cost and Diversity

Impact of varying Diverse Set Size

As shown in Figure 5.8(b) both Adaptive and Static Model schemes perform less number of operations as compared to Greedy. However cost of the Static Model scheme is upto 15% higher as compared to Adaptive model. This is because the static model fails to adjust for queries with different data distributions and hence the diverse results are located by examining most of the query results. Figure 5.8(a) shows that both schemes locate diverse subsets with diversity comparable to the subsets located by Greedy heuristic.

Impact of varying γ

In this experiment we focus on the impact of varying threshold parameter γ, that defines the acceptable difference ratio between the diversity values predicted by model and the actual diversity values. We compare the performance of Adaptive-FF-no-cache scheme against Greedy heuristic and Static Model approach. Figure 5.9(a) shows that both Greedy and Static Model remains unaffected by the change in γ. However as shown in Figure 5.9(a), as the threshold value is relaxed the adaptive model gets stable earlier and the predicted diversity values are used to locate diverse results earlier. This can be seen in higher cost savings as the value for γ increases. The savings in cost increase from 8% to 37% as the value of γ changes from 0.02



(a) Cost (b) Cost

FIGURE 5.10: Impact of θ on Cost and Diversity

to 0.054. However these savings in cost are at the expense of decrease in the diversity values of diverse subsets generated using adaptive model built with higher values of γ as shown in Figure 5.9(b). For $\gamma = 0.05$ the loss in diversity is upto 10%. The value of $\gamma = 0.03$ provides a good balance between cost savings and quality of diversification.

Impact of varying θ

In this experiment the performance of Adaptive-FF-no-cache scheme is compared against Greedy heuristic and Static model. The threshold value θ determines how close the diversity value of a subset should be from the predicted diversity to be acceptable. Figure 5.10(a) shows that as the threshold value is relaxed the cost savings for both the adaptive and static model as compared to the Greedy increase. For adaptive model these cost savings increase from 20% to 40% and for the static model the increase is from 5% to 30%. The diversity values decrease upto 12% with increase in the threshold value θ as shown in Figure 5.10(b). This is due to the fact that results far from predicted diversity value are also included in the diverse subset. If the user prefer higher cost savings at the expense of slight decrease in quality of diversification then higher values of θ are suitable, however if quality of diversification is more important that θ should be kept below 0.02.

5.4.2 Impact of using cached diverse results

In this experiment we evaluate the impact of using cached diverse results from overlapping queries. In particular, we compare the performance of scheme that employs adaptive regression model without cached results against the schemes that use adaptive model with cached results.

We have categorized this set of experiments into four sections by varying different experimental parameters.

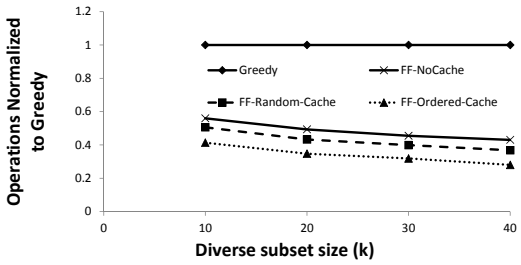


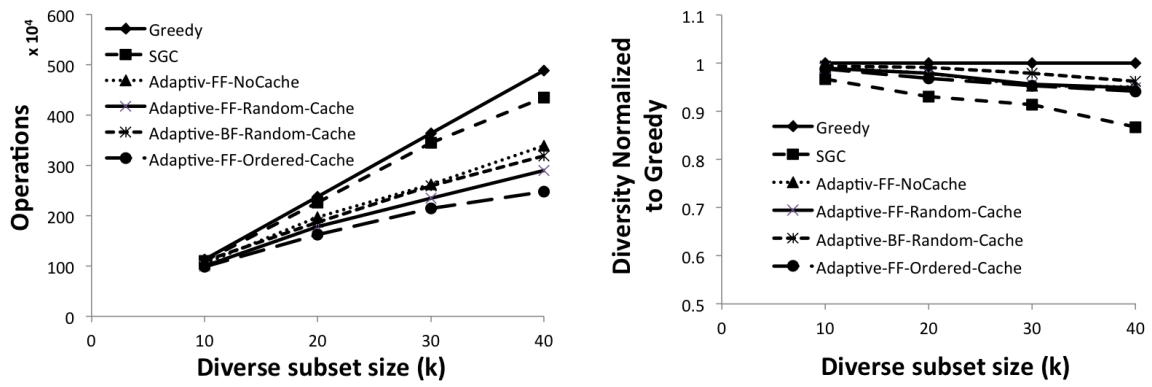
Figure 5.11: Impact of cache without model

Impact of Cached results in the absence of a Regression Model

In this experiment we have focused on the impact of using cached diverse results in the absence of a regression model. We use a hypothetical setting that serves as a yard stick to measure the effectiveness of using cached diverse results in locating the diverse results for the current query. Thus we assume the diverse subset for the current query is already evaluated using Greedy heuristic. Thus the actual diversity of the subset S_c^i is known for all the values of i, $2 \leq i \leq k$. These diversity values are used in each iteration i to locate the result that when added to S_c^{i-1} gives the same diversity value as S_c^i. We compare the performance of First-Fit and Best-Fit diversification schemes. As shown in Figure 5.11 the alternative schemes using cached diverse results are able to generate S_c in approximately 15% less number of operations as compared to FF-Nocache scheme that does not use any cached results. Also among different schemes FF-cache-ordered as discussed in section 5.2.4 performs best in terms of cost. Further details on ordering cached results are given in section 5.4.3

Impact of varying Diverse Set Size

In this experiment, we report on the impact of the required number of diverse results k. Figure 5.12(a) shows the number of operations (i.e., cost) performed by Greedy, SGC and AdoR schemes. As the figure shows, as the value of k increases, the cost increases for all schemes. AdoR, however, is performing up to 50% less operations than Greedy. Meanwhile SGC reduces diversification cost by only 10% as compared to Greedy. This is due to the fact that SGC utilizes only limited number of cached diverse results that are obtained from only one previous overlapping query. Among the two variants of AdoR, Adaptive-FF (first fit diversification) performs better in terms of cost as compared to Adaptive-BF (best fit diversification). In particular, FF-Random-Cache reduces the cost by up to 10% compared to BF-Random-Cache.



(a) Cost (b) Diversity

FIGURE 5.12: Impact of varying k on Cost and Diversity

This is clearly because First-Fit scheme terminates the search for an optimal result earlier without having to evaluate all the candidate results.

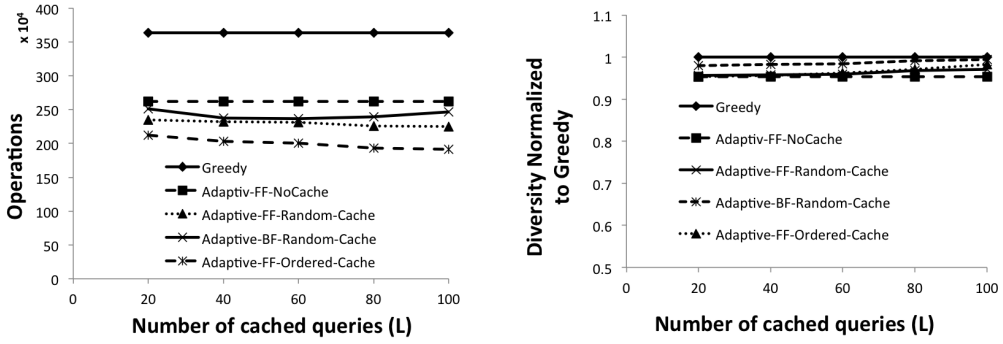
Further, Figure 5.12(a) also shows that between the two variants of Adaptive-FF scheme, the Adaptive-FF-Ordered-Cache performs upto 9% less operations as compared to Adaptive-FF-Random-Cache. Thus in terms of cost, Adaptive-FF-Ordered-Cache gives the best performance.

Figure 5.12(b) shows that the average diversity achieved by each scheme is decreasing with increasing the value of k. All AdoR schemes achieve comparable average diversity to Greedy. To highlight the benefits of AdoR in terms of achieved diversity, in Figure 5.12(b) the diversity values are normalized to the diversity values achieved by Greedy heuristic. As the figure shows, the maximum loss in diversity for AdoR is within only 5% compared to Greedy whereas it goes upto 14% for SGC. As the value of k increases the loss in diversity for SGC increases as it initializes diverse subset with higher number of overlapping results from the previous query without using any filtering.

Among the AdoR methods, Adaptive-BF performs better in terms of achieved diversity (Figure 5.12(b) because in each iteration, Adaptive-BF selects the one result providing the highest diversity if added to the diverse subset. Meanwhile, Adaptive-FF selects the first result that provides a diversity value within the threshold limit, thus introducing higher degree of approximation as compared to Adaptive-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%.

Impact of Cache Size

To study the impact of cache size in terms of number of cached queries, we generate 100 random queries across various user sessions. Then, we vary the number of cached queries L



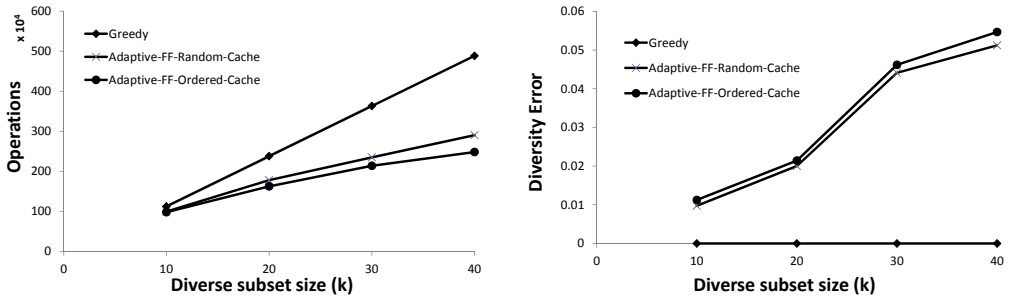
(a) Cost (b) Diversity

Figure 5.13: Impact of Cache Size on Cost and Diversity

from 20 to 100. Notice that in this experiment we are only evaluating the performance of AdoR schemes using cache, as SGC is not affected by the specified cache size. In terms of cost (i.e., number of operations), Adaptive-BF-Random-Cache exhibits an interesting pattern as shown in Figure 5.13(a). In that pattern, the cost of Adaptive-BF-Random-Cache decreases as the cache size increases up to a point, after which it starts increasing slightly. This is because at a moderate cache size, Adaptive-BF-Random-Cache 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, Adaptive-BF-Random-Cache still finds a cached result that is close to optimal, but it incurs a much higher cost in searching the rather large cache. Adaptive-FF-Random-Cache and Adaptive-FF-Ordered-Cache show a similar pattern, but are more resilient to the cached results as they terminate the search early. Adaptive-FF-Ordered-Cache has additional overhead of computing set distances from centroids of overlapping queries in cache. Thus as the cache size increases the difference in cost savings between Adaptive-FF-Random-Cache and Adaptive-FF-Ordered-Cache decreases to only 4%. Finally, Figure 5.13(b) shows that AdoR consistently achieves diversity comparable to Greedy algorithm when varying the number of cached queries.

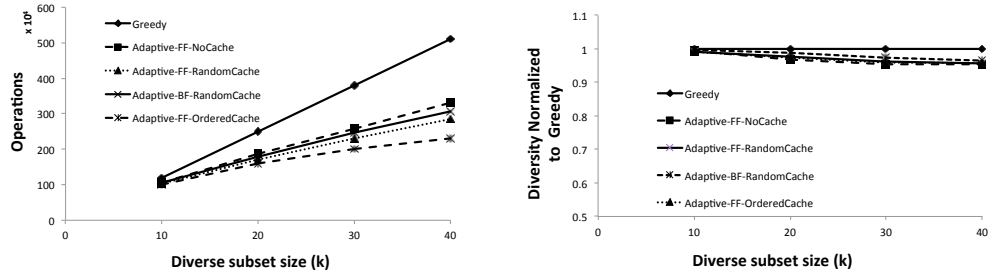
5.4.3 Impact of Ordering Cache

In this experiment we compare the performance of Adaptive scheme that uses cached results without any priority and the scheme that re-orders the cached results according to their distances from the diverse subset of current query. Figure 5.14(a) shows that among different variations of Adaptive scheme the Adaptive-FF-Ordered-Cache performs least number of operations. In particular, Adaptive-FF-Ordered-Cache performs 9% less operations as compared to Adaptive-FF-Random-Cache scheme. However the diversity values of the subsets generated



(a) Cost (b) Diversity

FIGURE 5.14: Impact of Ordering Cache (Cost)



(a) Cost (b) Diversity

FIGURE 5.15: Impact of varying k on Cost and Diversity (SDSS data set)

by both schemes are comparable as shown in Figure 5.14(b).

5.4.4 Results for SDSS Data Set

In this experiment, we report on the performance of AdoR scheme on the SDSS data set. Figures 5.15(a) and Figure 5.15(b) show that for the SDSS real data set, the performance of AdoR is comparable to its performance on the uniform data set. For different values of diverse subset size k, AdoR outperforms Greedy algorithm in terms of number of operations with negligible loss in achieved diversity.

5.4.5 Summary of Experimental Evaluation

In our experiments on both synthetic and real datasets, we have evaluated the effectiveness of model based diversification using caching in data exploration platforms. It has been shown that the regression model based greedy algorithm performs significantly less number of CPU operations for computing a diverse subset as compared to the greedy construction algorithm. The cost savings are further enhanced if cached diverse results from the previous queries are used. Among different methods proposed under AdoR scheme, the one using adaptive model

with ordered cache gives the best performance in terms of cost savings and quality of diversification. However, the number of cached queries needs to be selected carefully to keep the cost of searching within cache small. Similarly, when using regression model, the diversity of the subset of results is dependent on the user specified deviation threshold θ. It has been shown that if value of θ is kept below 0.02, it results in highly diverse subsets with cost savings up to 20%. All the AdoR variants consistently perform better than Greedy Algorithm for varying values of diverse subset size k.

5.5 Current Approaches to Multiple Query Diversification

In recent years several diversification approaches have been proposed as in [24, 30, 57, 65, 96, 99, 103]. Despite the considerable interest in diversification, most of the existing approaches consider diversification of a single query result. Recently, in [19] a new coverage based definition of diversity is formulated for generating a combined diverse subset that recovers the results of multiple queries. However, our proposed AdOr scheme computes a separate diverse subset, using a traditional content-based definition of diversity that represents each query result set. Also, unlike [19], in our problem setting the queries are generated sequentially at different intervals of time. Therefore, the query results are not available simultaneously for generating a combined solution.

The central element of our approach is the use of a probabilistic model to estimate the diversity value of a diverse subset for future iterations of Greedy algorithm. Lots of work related to approximate query processing in database community makes use of some form of probabilistic models (e.g., [26,79]). However, to the best of our knowledge this is the first work where model based approach is used in the post query processing to diversify the query results.

Data caching and pre-fetching have also been shown as important approaches for reducing the cost of exploration queries [50]. In particular, caching has been used for efficient computation of representative data in interactive data exploration. For instance, in [16] a series of refined queries are evaluated by appropriately exploiting the information generated during the execution of previous queries in order to return top-K results. In [76], a caching mechanism is proposed that helps reduce the cost of computing future dynamic skyline queries by caching the results of previous skyline queries. While different caching approaches have been used in the literature for efficient representative data extraction, each approach varies in its methodology on "what to cache" and "how to use the cache" depending upon the end objective. Two closely related works using caching for search results diversification are presented in [65] and [30].

Specifically in [65], cached distance computations are used to reduce the cost of Greedy heuristic whereas in [30], the idea of reusing already computed diverse results has been discussed. Especially, [30] extends the basic Greedy construction heuristic 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. Similarly, in our problem setting 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 to have common results, similar to two consecutive sliding windows in a data stream. The premise underlying the proposed Stream Greedy Construction scheme in [30] is that instead of re-evaluating all the k diverse results for each sliding window, the diverse subset of the current window is initialized using the diverse results from the previous window. The drawback of this approach is the assumption that every window is uniformly populated from the data space. Thus, it is assumed that the valid diverse results from the previous window are still diverse with respect to the new data in the current window. However, in many data stream scenarios data distribution across different windows can be quite different. Thus, in our work we selectively choose only those diverse results from the cache that are still diverse with respect to the result of the current query. In the section 5.4, we have compared the performance of Stream Greedy Construction (SGC) against our proposed schemes.

5.6 Summary

Search Results Diversification has emerged as an important representative data extraction technique for Interactive Data Exploration platforms. In order to reduce the overhead of Diversification in an already comutationally expensive exploration process, in this chapter, we have presented the *(Adaptive model based diversification AdOr* scheme. AdOr 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 an adaptive model-based approach that is particularly suitable for the efficient and effective diversification in IDE. AdOr 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.

Chapter 6

Diversity with few Regrets

As already mentioned in Chapter 2, emerging Data exploration platforms typically apply novel post-processing techniques on both the queries and their respective answers to provide users with guidance and insights. Generating representative data is one such technique that aims to provide meaningful summary of a potentially large query answer (e.g., [20, 31, 66, 87, 97]). In Chapters 3, 4 and 5, we have addressed the computational complexity of generating representative subsets having diversity as the only selection criteria. However, in many applications users may have some notion of preference along few dimensions whereas the coverage and diversity is desired along other dimensions. Hence, in this chapter we have focused on the complexity of the multi-objective function when generating representative subsets. Specifically, we have considered the problem of combining diversity with other user preferences. Among state of the art representative extraction techniques, Top-k and Skyline queries have gained much attention for capturing user preferences. In top-k, the user's preference is captured by means of a utility function over different dimensions of data, whereas in skyline, that preference is captured by applying the dominance property over those dimensions. Recently, regret minimization has been proposed as a practical alternative for both queries [66]. In regret minimization, a small representative set is generated by considering the universe of all possible utility functions. Hence, a user does not need to specify a specific utility function, as it is the case in top-k, but are still provided with a small and concise representative set, unlike the skyline query, in which the result can be arbitrarily large.

While incorporating diversity in top-k (e.g., [6, 41, 72]) and skyline (e.g., [47, 95]) queries have been studied in literature, objective function using both diversity and regret minimization criteria has not been considered. Therefore, in this work we have looked in to the problem of computing representative subsets with an objective of maximizing diversity and minimizing

TABLE 6.1: Car Database

Car	MPG	HP	Weight	Height
p_1	51	134	1760	52.4
p_2	40	110	2945	48.8
p_3	41	191	1875	54.3
p_4	35	198	2050	56.3
p_5	30	140	2215	50.6

regret. For instance, assume a user who user might have some notion of preference associated with some dimensions of the data, while other dimensions are neutral. In that case, it is desired to select representatives that: 1) minimize regret over the preference dimensions (i.e., high utility), and 2) maximize diversity over the neutral dimensions (i.e., low redundancy).

For example, consider a tourist visiting downtown Melbourne for SIGMOD and is looking for few restaurants to try during her visit. That user might have some preference for restaurants with low price and high rating. At the same time, she might not want all restaurants to be cluttered in one location so that she gets to see more of the city during her short visit.

To capture that tradeoff between preference and coverage, we propose a novel scheme called ReDi, which aims to generate representative data that balance the tradeoff between regret minimization and diversity maximization. Our proposed scheme ReDi is based on a hybrid objective function formulated as the linear weighted combination of the diversity and regret objectives. To that end, ReDi incorporates two alternative novel algorithms that are based on two different algorithmic design approaches. In particular, we propose the ReDi-Greedy algorithm, which is a constructive based heuristic, and we also propose ReDi-SWAP, which is a local search based algorithm. Further, we study the tradeoff those two algorithms exhibit in terms of efficiency and effectiveness.

The rest of this chapter is organized as follows. We present preliminary concepts on regret minimization in Section 6.1. Next we formulate our hybrid objective function in Section 6.2, and present our ReDi scheme in Section 6.3. Our evaluation testbed and results are reported in Section 6.4. We summarize in Section 6.5.

6.1 Minimizing Regret

In the presence of user preference over some of the data dimensions, representatives are typically selected based on those preferences. There has been lot of work in literature that addresses

the problem of personalizing user search based on user preferences. For instance, in [87] the user preferences are expressed as user choice that holds under specific context. The top-k results to the keyword query are ordered on the basis of the user preferences. More commonly used approach for specifying user preference in top-k queries is through utility functions. For instance, in a *top-k* query, the user specifies a utility function g, such that the utility of a D-dimensional point p is given by $g(p)$. Hence, the top k points based on their values under that utility function are then selected as representatives. Alternatively, in a *skyline* query, any point for which there is no other point with better values in all D dimensions is selected. However, in a *top-k* query the user is required to precisely define a utility function, which is often unknown, whereas in a *skyline* query there is no control on the size of the output, which can be arbitrarily large. Such drawbacks motivated the recent regret minimization methods [66].

Particularly, the goal of regret minimization is to select a subset of size k, which minimizes the maximum regret ratio for any class of utility functions. This captures how disappointed any user could be had they seen k representative tuples instead of the whole database [66]. Specifically, for a certain user with a utility function $g \in G$ and a subset of points $S \subseteq P$, the regret of that user is $max_{p \in P} g(p) - max_{p \in S} g(p)$, where $max_{p \in P} g(p)$ is the maximum utility if the user saw the entire database P, whereas $max_{p \in S} g(p)$ is the maximum utility if the user saw only the representative set S. Accordingly, given a class of utility functions G, the maximum regret ratio of a subset S, denoted $rr_P(S, G)$, is defined as:

$$rr_P(S, G) = sup_{g \in G} \frac{max_{p \in P} g(p) - max_{p \in S} g(p)}{max_{p \in P} g(p)}$$

In this work, we restrict G to be the class of linear functions with positive weights, as in [66]. Given that restriction, consider again the dataset of table 1 and suppose there is some notion of preference associated with attributes MPG and HP. For simplicity, further assume $G = \{g_{\{0.2,0.8\}}, g_{\{0.4,0.6\}}, g_{\{0.6,0.4\}}, g_{\{0.8,0.2\}}\}$ where: $g_{\{x,y\}}(MPG, HP) = MPG.x + HP.y$ If $S = \{p_1, p_2\}$, then $rr_P(S, G) = 0.29$. If, however, $S = \{p_3, p_4\}$, $rr_P(S, G) = 0$. However, if the goal is to select a set of 2 cars that is diverse in weight and height, then the set $S = \{p_1, p_4\}$ is the most diverse under the diversity definitions formulated in the previous section.

In general, computing the value $rr_P(S, G)$ for a class G is based on finding the "worst" point. That is, the point that contributes to the currently perceived maximum regret ratio after S points have been selected. Hence, that computation is achieved by running a linear program to compute $rr_{S \cup \{p\}}(S, G)$ for each $p \in P \backslash S$ to find the one point p' that is responsible for the current maximum regret ratio [66]. Hence, $rr_P(S, G) = rr_{S \cup \{p'\}}(S, G)$, where $rr_{S \cup \{p\}}(S, G)$ is

evaluated using the following linear program [66]:

$$\begin{aligned} & \max \ \ x \\ \text{s.t.} \ \ & \sum_{i=1}^{D}(p[i]-p'[i])v[i] \geq x \ \ \forall p' \in S \\ & \sum_{i=1}^{D} p[i]v[i] = 1 \\ & v[i] \geq 0 \ \ \forall i \leq D \\ & x \geq 0 \end{aligned} \tag{6.1}$$

Notice that in the linear program above, each possible utility function is represented by a vector v in D dimensions with non negative coordinate values. Hence, the linear program finds the vector v that maximizes the regret ratio of S relative to $S \cup p$, and the value x returned by the linear program is precisely $rr_{S\cup\{p\}}(S, G)$.

Similar to maximizing diversity, the problem of regret minimization has also been shown to be NP-hard [20]. Hence, several greedy heuristics have been proposed to find near-optimal solutions for regret minimization [66]. Those heuristics are based on the Greedy algorithm proposed in [66], in which S is constructed iteratively, where in each iteration, the point that contributes to the maximum regret ratio is selected and added to S, until k points are selected. The pseudo code for this algorithm is given in algorithm 10, where $rr_{S\cup\{p\}}(S)$ is evaluated using the above linear program.

6.2 Combining Diversity and Regret

Clearly, the methods for diversity maximization and regret minimization compute different representative sets that optimize their respective objective functions. For example, consider Figure 6.1(a), which shows a small data set consisting of 12 points $p_1, p_2, ..., p_{12}$, each with 7 dimensions $A_1, A_2, ..., A_7$. For each point p_i, the normalized attribute value for each of its 7 attributes is shown on the y-axis. Further, assume there is some notion of preference associated with the first four attributes A_1 to A_4 (e.g., the higher the value, the better), whereas there is no such notion defined for the remaining three attributes $A5$ to A_7. Hence, in generating a representative set S, it is desired to select points that: i-minimize regret over the first four dimensions (i.e, high utility), and ii-maximize diversity over the last three dimensions (i.e., low redundancy). Figure 6.1(b) shows a set S of size 5 points selected by the Greedy algorithm for regret minimization (called *Reg-Greedy* hereafter). The figure shows that selected points have high values under one or more of the first four attributes (i.e., high utility), but have very similar

Algorithm 10 Regret Greedy Algorithm

Input: A set of D dimensional points $P = \{p_1, p_2, ..., p_n\}$ and an integer k for output size

Output: A subset of P of size k denoted by S

$S \leftarrow \{p\}$ such that $p = \underset{p_i \in P}{argmax}\, p_i[1]$

for $i = 1$ to $k - 1$ **do**

  $r* = 0$

  $p* = null$

  for all $p \in P \setminus S$ **do**

    if $r* < rr_{S \cup \{p\}}(S)$ **then**

      $r* \leftarrow rr_{S \cup \{p\}}(S)$

      $p* \leftarrow p$

    end if

  end for

  $S \leftarrow S \cup \{p*\}$

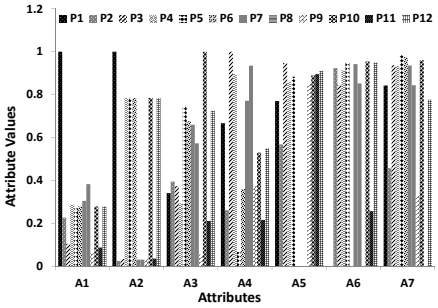
end for

values in the last three attributes (i.e., high redundancy). Alternatively, Figure 6.1(c) shows the set S selected by the Greedy algorithm (Algorithm 1) for diversity maximization (called *Div-Greedy* hereafter). In contrast to Figure 6.1(b), Figure 6.1(c) shows that the selected points have diverse values under the last three attributes (i.e., low redundancy), but miss the user preference for higher values along the first four dimensions (i.e., low utility). Hence, neither of the two algorithms manages to achieve both high utility and low redundancy at the same time.

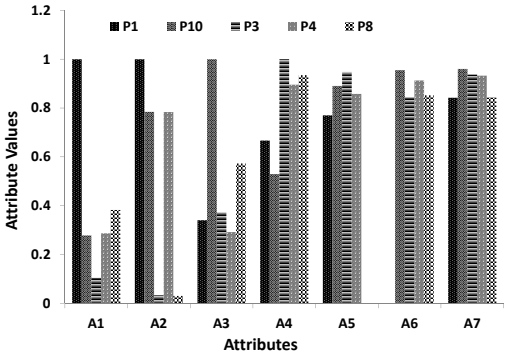
To capture the conflict illustrated in the previous example, we utilize a hybrid function that considers both diversity and regret. Specifically, for a subset $S \subseteq P$, an objective function is formulated as the linear weighted combination of the scaled diversity and regret objectives, which is defined as:

$$\mathcal{F}(S, G, P, \lambda) = \lambda \frac{\sum_{i=1}^{k} \sum_{j>i}^{k} d(p_i, p_j)}{\underset{p_i, p_j \in P}{max}\, d(p_i, p_j)} + (1 - \lambda) \frac{k(k-1)}{2} (1 - rr_P(S, G)) \tag{6.2}$$

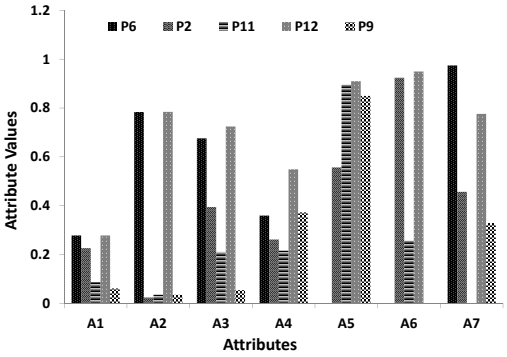
where $k = |S|$, and λ is the weight parameter for balancing the tradeoff between diversity and regret, such that $0 \leq \lambda \leq 1$. Notice that the sum of the pairwise distances is divided by the maximum distance between any pair of points in the whole set so as to normalize the value of the distances between 0 and 1. Similarly, the regret objective is scaled up by $\frac{k(k-1)}{2}$ which is the total number of pairs considered in the sum of the distances for diversity.



(a) All Records



(b) Regret minimized set (k=5)



(c) Diverse set (k=5)

FIGURE 6.1: Impact of Regret Minimization and Diversification

When diversity and regret are associated with different dimensions, our objective function defined above is easily modified accordingly. In particular, if $V_{div} \subseteq \{A_1, ..., A_D\}$ represents the dimensions designated for diversification and $V_{reg} \subseteq \{A_1, ..., A_D\}$ represents the dimensions on which regret minimization is required, our objective function is reformulated as:

$$\mathcal{F}(S, G, P, \lambda, V_{div}, V_{reg}) = \lambda \frac{\sum_{i=1}^{k} \sum_{j>i}^{k} d(p_i, p_j, V_{div})}{\max\limits_{p_i, p_j \in P} d(p_i, p_j, V_{div})} + (1-\lambda)\frac{k(k-1)}{2}(1 - rr_P(S, G, V_{reg}))$$

Hence, the goal is to find a set S^*, which balances the tradeoff between diversity and regret by maximizing the objective function $\mathcal{F}$ defined above. Formally:

$$S^* = \underset{S \subseteq D, |S|=k}{argmax} \mathcal{F}(S, G, D, \lambda, V_{div}, V_{reg}) \tag{6.3}$$

Note that in this work we focus on the case where diversification and regret minimization are desired on different dimensions, which means $V_{div} \cap V_{reg} = \phi$. Referring back to the sample data in table 6.1, assume $V_{div} = \{Weight, Height\}$, $V_{reg} = \{MPG, HP\}$, and it is required to select a representative set S of size 2 (i.e., $k = 2$). For that data, $S = \{p_1, p_3\}$ is the the set which minimizes the maximum regret ratio in MPG and HP, but provides very low diversity in Weight and Height. On the other hand, $S = \{p_2, p_4\}$ maximizes the diversity in Weight

and Height, but has a very high maximum regret ratio in the first 2 dimensions. Giving equal weight to both diversity and regret by setting $\lambda = 0.5$ in $\mathcal{F}$, we find that the subset of size $k = 2$ which maximizes $\mathcal{F}$ is $S = \{p_1, p_4\}$, where the points in S provide the desired balance between minimizing regret and maximizing diversity.

6.3 The ReDi Scheme

In this section, we present our ReDi scheme for balancing the tradeoff between minimizing regret and maximizing diversity. In particular, we present two algorithms that aim to achieve that goal as it is captured by the hybrid function presented in Section 6.2. Towards this, we present two algorithms: ReDi-Greedy, and ReDi-SWAP, and study the tradeoff they exhibit in terms of both efficiency and effectiveness.

6.3.1 ReDi-Greedy

ReDi-Greedy follows the same general design adopted by the class of constructive algorithms for solving different optimization problems, including those for regret minimization, and diversity maximization. In constructive algorithms, a final solution is achieved incrementally in steps, where in each step a local decision is made based on some criteria, where the choice of such criteria depends on the target optimization problem. ReDi-Greedy also constructs the result set S iteratively by selecting a new point in each iteration, where the criteria for selecting such point is based on the objective function $\mathcal{F}$ defined in Section 6.2. Particularly, based on $\mathcal{F}$, it is desirable to select in each iteration a point that can potentially contribute the most to decreasing the regret of the current set S and also increasing its diversity. In order to locate such point, ReDi-Greedy employs a heuristic priority function, where each point in P that is not in S is assigned a score, which is defined as follows:

$$Score(p, S, V_{div}, V_{reg}) = \lambda \frac{SetDist(p, S, V_{div})}{max_{p_i \in P \setminus S} SetDist(p_i, S, V_{div})} + (1 - \lambda) \frac{rr_{S \cup \{p\}}(S, G, V_{reg})}{rr_P(S, G, V_{reg})}$$

Where SetDist(p,S), between a point p and a set of points S is defined based on its distance from the points in S, as: $SetDist(p, S) = \frac{1}{|S|} \sum_{p_j \in S} d(p, p_j)$.

Thus each candidate point p is assigned a score, which is the weighted sum of its set distance from S and the maximum regret ratio of set S with respect to p. Both the set distance and regret components are normalized by dividing the first term by the maximum set distance of S

Algorithm 11 ReDi-Greedy

Input: A set of D dimensional points P, a set V_{div}, a set V_{reg}, an integer k and λ

Output: A subset of P of size k denoted by S

$S \leftarrow \{p\}$ such that $p = \underset{p_i \in P}{argmax}\, p_i[1]$

for $i = 1$ to $k - 1$ **do**

  $maxScore = 0$

  $p* = null$

  for all $p \in P \setminus S$ **do**

    if $maxScore < Score(p, S, , V_{div}, V_{reg})$ **then**

      $maxScore \leftarrow Score(p, S, V_{div}, V_{reg})$

      $p* \leftarrow p$

    end if

  end for

  $S \leftarrow S \cup \{p*\}$

end for

TABLE 6.2: $Score()$ vs. $\mathcal{F}()$ at $S = \{p_2\}$

Point	p_1	p_3	p_4	p_5
$Score(p)$	0.99	0.96	1	0.528
$F(S \cup p)$	0.627	0.726	0.724	0.395

and the second component by the maximum regret ratio of S with respect to the whole set P. The score of each candidate point p thus measures the potential contribution of p in increasing the objective function value for set $S \cup p$. Therefore in each iteration the point with maximum score is added to set S (the pseudocode for ReDi-Greedy is presented in algorithm 11).

Although ReDi-Greedy is very efficient in practice, there is no guarantee that the point $p*$ picked in each iteration is actually the best point (i.e., local minimum) for the objective function $\mathcal{F}$ given the current representative set. This is because the point $p*$ that has the highest score may not necessarily be the one that improves the combined function value the most. Consider again the example in table 6.1 and assume that our current set $S = \{p_2\}$ and $\lambda = 0.5$. Table 6.2 lists the scores of each of the other points based on the above scoring function as well as the actual function values obtained by adding that particular point to the current S. As the table shows, the highest scoring point, p_4 does not yield the highest function value when added to S, which is given by p_3.

To address the limitations of ReDi-Greedy, next we present the ReDi-SWAP heuristic, in

Algorithm 12 ReDi-SWAP Algorithm

Input: A set of D dimensional points P, a set V_{div}, a set V_{reg}, an integer k and λ

Output: A subset of P of size k denoted by S

$S \leftarrow regretGreedy(P, k, V_{reg})$

$S' \leftarrow S$

for all $p \in P \setminus S'$ **do**

$S_{temp} \leftarrow S$

$p^* = \underset{p \in P \setminus S'}{argmax} \sum_{p_i \in S} d(p, p_i, V_{div})$

$S' \leftarrow S' \cup \{p^*\}$

for all $p' \in S$ **do**

if $F(S_{temp}, G, P, \lambda, V_{div}, V_{reg}) <$

$F(\{S \backslash p'\} \cup p^*, G, P, \lambda, V_{div}, V_{reg})$ **then**

$S_{temp} = \{S \backslash p'\} \cup p^*$

end if

end for

if $F(S_{temp}, G, P, \lambda, V_{div}, V_{reg}) > F(S, G, P, \lambda, V_{div}, V_{reg})$ **then**

$S \leftarrow S_{temp}$

end if

end for

which the selection of points is based on the actual improvement in the value of the objective function $\mathcal{F}$ instead of the expected improvement.

6.3.2 ReDi-SWAP

The ReDi-Greedy algorithm presented in the previous section is of the constructive type. That is, it starts without a representative set and incrementally constructs it by adding one point at a time. To the contrary, ReDi-SWAP presented in this section falls under the local search type of algorithms. In general, a local search algorithm starts out with a complete initial solution and then attempts to find a better solution in the neighborhood of that initial one. Like constructive algorithms, local search algorithms are also widely used in solving optimization problems including generation of representative data. For instance, the SWAP local search method has been utilized to maximize diversity [31, 97], and in this paper, we further expand into our ReDi-SWAP method for balancing the tradeoff between regret and diversity.

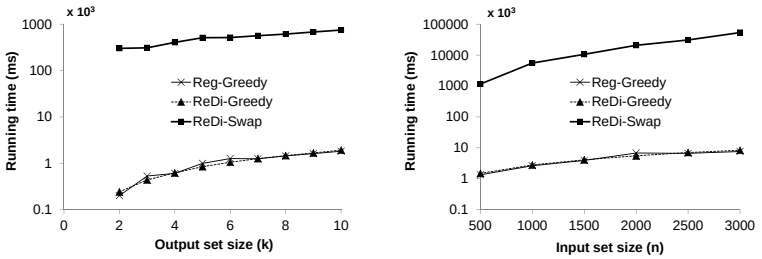
The basic idea underlying ReDi-SWAP is to start with an initial set S of size k and then

iteratively modify the set S in order to improve the value of the objective function $\mathcal{F}$. One of the main design criteria in local search algorithms is the choice of the initial solution. In ReDi-SWAP, we opt to initialize S with the k points which minimize the regret ratio as selected by the traditional Reg-Greedy algorithm [66]. Another important criterion is the neighborhood definition of local search together with the search process. Since we initialize ReDi-SWAP with the regret minimization set, the neighborhood of the local search is explored based on the second component of our objective function (i.e., diversity). In particular, the points in $P \backslash S$ are visited according to their distance from S, such that the points with higher distance are visited first. Accordingly, in each iteration, from the set of points that haven't yet been visited, given by $P \backslash S'$, the point with the highest distance from the current set S is tested, denoted by $p*$. This point is tested against each point in S by removing that point p' from S and adding $p*$ to the set. After going through all the points in S one by one, the swap that results in the highest function value is made if it also is an improvement over the set S before the swaps were made. The intuition is to replace the point in S which contributes the least to its diversity with a point from $P \backslash S$ which would improve that diversity while at the same time maintaining regret very close (or equal) to that of S, which is easily captured by evaluating the hybrid objective function $\mathcal{F}$.

Notice that for any candidate point p, the decision made by ReDi-SWAP is based on evaluating the objective function $\mathcal{F}$ if p is selected to join S. Evaluating $\mathcal{F}$ requires $O(n)$ calls of the linear program to compute the maximum regret ratio and $O(k^2)$ distance computations to calculate the diversity of set S. Under ReDi-Greedy, however, the decision on a candidate point p is based on assigning a priority value (i.e., score) to p. For that decision, it is enough to evaluate the regret contributed by p but not the actual regret achieved when p is added to S, which requires one call of the linear program. Hence, in total, ReDi-Greedy performs $O(nk)$ calls to the linear program, whereas ReDi-SWAP performs $O(n^2k)$. The tradeoff between the efficiency and effectiveness provided by these two algorithms is evaluated experimentally in the next section.

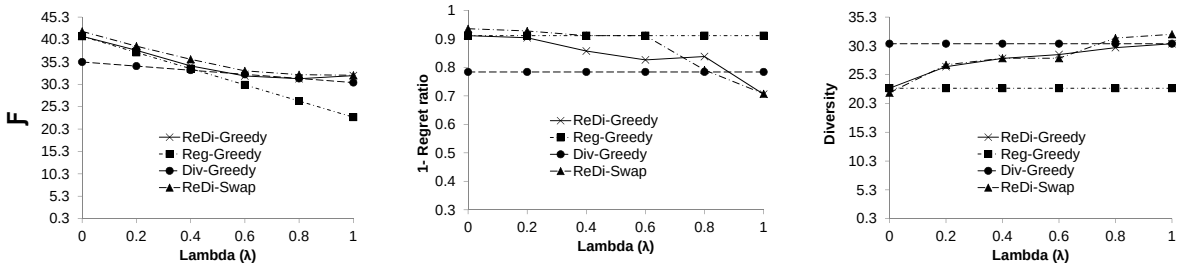
6.4 Experimental Evaluation

We perform several experiments to evaluate the performance of the different schemes discussed in this paper, namely: i) Div-Greedy ii) Reg-Greedy, iii) ReDi-Greedy, and iv) ReDi-SWAP. In particular, we compare those algorithms in terms of effectiveness, which is measured in terms of the objective function $\mathcal{F}$, and efficiency, which is measured as total time spent in executing



(a) (b)

Figure 6.2: Cost of different methods in terms of running time



(a) Function Value (b) Regret Ratio (c) Diversity

Figure 6.3: Impact of λ

the linear program optimization.

Experiments are conducted on a simple synthetic 10-dimensional dataset of size 3K tuples, in which the attribute values of each dimension are generated uniformly in the range [0–1]. Table 6.3 summarizes the database settings together with the other parameters considered in our evaluation.

Figure 6.3(a) shows the impact of λ on the value of the objective function $\mathcal{F}$. As shown in the figure, ReDi-SWAP provides the highest values for $\mathcal{F}$ followed by ReDi-Greedy. Meanwhile, Reg-Greedy and Div-Greedy are both oblivious to any tradeoff between regret and diversity, which translates into lower $\mathcal{F}$ values. This is further illustrated in Figures 6.3(b) and 6.3(c). Figure 6.3(b) shows the happiness measure of a set S computed as 1-regret ratio(S). As expected, the figure shows that Reg-Greedy provides the highest happiness, whereas Div-Greedy provides the lowest. The figure also shows that ReDi-Greedy and ReDi-SWAP provide slightly lower happiness than Reg-Greedy but much higher than Div-Greedy. The same behavior is exhibited in Figure 6.3(c) with respect to diversity.

The improvement in the $\mathcal{F}$ value provided by ReDi-SWAP over ReDi-Greedy (as shown in Figure 6.3(a)) comes at the expense of increasing the processing time, as shown in Figure 6.2, which shows the time spent in executing the LP optimizations and distance computations. This is to be expected since ReDi-SWAP has higher complexity than ReDi-Greedy, as discussed

TABLE 6.3: Evaluation Setting.

Parameter	Range	Default
Number of Dimensions (D)	2–10	10
Database Size	3k	300
Representative Set Size (k)	2–10	5
Weighting Factor (λ)	0–1	0.5
Number of Regret Dimensions $\|V_{reg}\|$	2–8	5
Number of Diversity Dimensions $\|V_{div}\|$	2–8	5

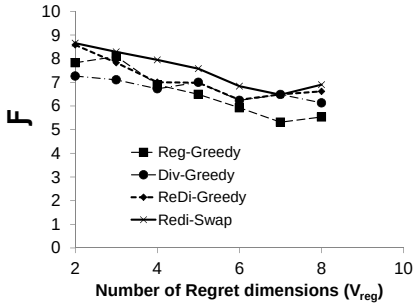


FIGURE 6.4: Impact of dimensions split

in the previous section. eatTo further illustrate the tradeoff between ReDi-Greedy and ReDi-SWAP, we compare them relative to the optimal solution in Figure **??**. The figure shows the $\mathcal{F}$ provided by each of the two algorithms normalized to the optimal exhaustive solution for a small dataset ($n = 30$) while varying k. As shown in the figure, the $\mathcal{F}$ value of ReDi-SWAp is 95-100% of the optimal, whereas ReDi-Greedy achieves around 85-90% of the optimal.

In all the previous experiments, the 10 dimensions were split evenly between regret and diversity (i.e., $|V_{reg}| = 5$ and $|V_{div}| = 5$), as in the default setting. In this experiment, we measure the impact of changing that split as shown in Figure 6.4. The figure shows the $\mathcal{F}$ value as we increase the number of dimensions considered for regret minimization, or equivalently decreasing the number of dimensions for diversity maximization. As the figure shows, the overall trend for all algorithms appears to be a decrease in $\mathcal{F}$ value as $|V_{reg}|$ is increased. This is consistent with results in literature on the impact of dimensions on regret minimization and diversity maximization. In particular, maximum regret ratio has been observed to worsen, or increase, with the increase in dimensionality whereas diversity increases with increase in dimensionality. Both of these observations are consistent with the trend in figure 6.4, where the $\mathcal{F}$ value decreases as the number of dimensions for regret increase and those for diversity are reduced.

All the previous results indicate that ReDi-SWAP is a more effective scheme than ReDi-Greedy. However, that improvement in effectiveness comes at the expense of a higher computational cost. Such a high cost would be prohibitive when applying ReDi-SWAP on large databases. Hence, from a practical standpoint, ReDi-Greedy presents a more attractive solution due to its scalability, while at the same time providing $\mathcal{F}$ values close to those achieved by ReDi-SWAP.

6.5 Summary

We considered the problem of simultaneous regret minimization and diversity maximization on multi-dimensional data where both these objectives are desired on different dimensions. To that end, we captured the tradeoff between those two objectives by means of a hybrid function, which also forms the basis of our new ReDi scheme. As part of our ReDi scheme, we proposed two alternative solutions that fall under two major classes of optimization algorithms, namely: local search optimization, and constructive optimization. Our experimental evaluation studies the efficiency and effectiveness of those algorithms under different parameters.

Chapter 7

Conclusions and Future Work

The goal of this thesis was the design, implementation and evaluation of algorithms and schemes for the scalable diversification of search results in data exploration platforms. Next, in Section 7.1, we summarize our contributions towards that goal and in Section 7.2, we describe directions for future work.

7.1 Summary of Contributions

We have addressed the challenging problem of search results diversification in data exploration platforms. While diversification, like other data summarization techniques, provides users with quick insights into the query answers, it adds additional complexity to an already computationally expensive data exploration task. Hence, in order to present users with diverse search results at minimum additional cost we have proposed various scalable diversification schemes in this thesis as summarized below.

In Chapter 3, we presented the *Progressive Data Diversification (pDiverse)* scheme for efficient diversification of high dimensional large datasets. The main idea underlying pDiverse is to utilize *partial distance* computation to reduce the amount of CPU and I/O incurred during query diversification. In a traditional approach where diversification is decoupled from query processing, our scheme allows to quickly detect and prune those data points in the query result that cannot be included in the final diverse set. The early pruning of those points provides significant reduction in the CPU cost required for distance computations. Moreover, we propose integrating data diversification with query processing, which enables pushing down partial distance computation closer to the raw data, and hence provides pruning at the data storage layer. This allows for saving the I/O costs incurred in accessing those pruned values, especially

in vertically partitioned data (i.e., column-stores). In particular, *Column-store* systems vertically partition a database into a collection of individual columns, which are stored separately. Hence, instead of reading all the attribute values of all the data points accessed by a query, pDiverse leverages a column-based storage to selectively read only those attribute values of the unpruned data points and save I/O cost. Our extensive experimental evaluations on real and synthetic data sets illustrate the benefits achieved by pDiverse.

Besides processing large volumes of data, another key feature of Data Exploration systems is the ability to host multiple users, executing multiple queries in parallel sessions. Meanwhile, delivering near real time performance remains an essential requirement for Data Exploration platforms so that to match the intrinsic nature of interactive and iterative data exploration to ensure user satisfaction. Hence, for scalable diversification of multiple queries within and across users sessions, we have proposed two schemes for diversification of multiple query results in Chapter 4 and 5.

In Chapter 4, we have proposed the DivM scheme for concurrent diversification of simultaneous queries across user sessions. In particular, DivM leverages the natural overlap in search results in conjunction with concurrent diversification of those results using partial aggregation techniques. This enables DivM to provide the same quality of diversification as that of the sequential methods, while significantly reducing the processing costs. We further generalize and extend the DivM scheme to exploit various approximation techniques that provide orders of magnitude reductions in processing cost, while maintaining a quality of diversification comparable to that of near optimal schemes. Our extensive experimental evaluation on both real and synthetic data sets shows the scalability exhibited by our proposed scheme under various workload settings, and the significant benefits it provides compared to existing methods.

For Sequential Diversification of multiple queries within a user session, we have presented an efficient diversification scheme in Chapter 5. Our proposed scheme, called AdOr, relies on two main interrelated components, namely: 1) an adaptive model-based diversification method, and 2) an order-based caching scheme. In particular, AdOr employs an adaptive model based diversification method to estimate the diversity of a diverse subset and hence selects diverse results without scanning all the query results. In order to further expedite the diversification process, AdOr employs an order-based caching scheme to leverage the overlap between sequence of data exploration queries. We conducted extensive experimental evaluation on real and synthetic data sets, which compare the performance of multiple diversification schemes and illustrate the benefits achieved by AdOr.

Besides the computational complexity of diversification in Data Exploration, we have also

addressed the complexity of the objective function when considering diversification with other criteria. In contrast to the current approaches that focus on one objective function for generating representative data, we have addressed the problem of combining diversity with regret minimization. In particular, we address the scenario where users typically have some prespecified preferences over some dimensions of the data, while expecting good coverage over the other dimensions. Our proposed scheme called ReDi, which is presented in Chapter 6, aims to generate representative data that balance the tradeoff between regret minimization and diversity maximization. ReDi is based on a hybrid objective function that combines both regret and diversity. Additionally, it employs several algorithms that are designed to maximize that objective function.

7.2 Future Work

In this section, we propose possible directions for the future work.

7.2.1 Distributed Data Diversification

In the future, we propose investigating the diversification problem in a distributed setting where data is stored across various nodes. The current centralized diversification algorithms require all the relevant data on a single node. However, in a distributed environment, transfer of such data from various nodes to a central node is a challenging task, either because of privacy concerns or due to network bandwidth limitations, or because of the huge amount of distributed data. Hence, bandwidth efficient distributed diversification algorithms are needed for both vertically (i.e., different columns/dimensions are stored on different nodes) and horizontally (e.g., rows of data are distributed across different nodes) partitioned datasets.

For vertically partitioned data, we can extend our pDiverse scheme as presented in Chapter 3. The pDiverse scheme computes diverse subsets of results by progressively visiting each dimension. Since, all data is available on the same node, the I/O cost of accessing each data block for all dimensions is the same. Similarly, there are no communication delays involved. However, in a distributed environment, the cost of accessing data from each node may vary drastically. Hence, the I/O cost and communication delay for each dimension need to be incorporated in the dimension ordering function. Moreover, random data access to retrieve few values on demand can be an expensive operation when data is stored remotely. Hence, new

data reduction and pruning methods need to be investigated for minimizing the communication delays as well as random data accesses when diversifying results over distributed datasets.

For horizontally partitioned datasets, the DivM scheme presented in Chapter 4 can be further extended. DivM evaluates diverse subset by partitioning the underlying query result set into overlapping and non-overlapping sets. Each set is processed independently to select the local optimum result with maximum distance from already selected diverse results. Those local optimum results are later compared to select the global optimum result. The independent processing of each partition by DivM can be utilized for parallel diversification of results across multiple data nodes. However, as the diverse subset is updated after each iteration, it needs to be shared with each data node. Such sharing can be unauthorized in some situations where data privacy policies are in place. In those situations, new novel methods are needed for evaluating partial diverse subsets at each node and then merging them to compute a global diverse subset.

7.2.2 Diversification of Multiple Queries over Data Streams

In this thesis, we have addressed the problem of multiple diversification queries over static datasets. However, there is a growing increase in the number of real time applications that need to handle continuous queries over streaming data e.g., retail chain transactions, tracking moving objects, monitoring online network behaviours, and sensor data processing. Note that unlike static data, stream data objects hold a property related to time, and many applications often consider that older data objects are less important. Thus, often a time-based sliding window is assumed. In addition, users are also allowed to specify a region of interest. Then, the diversification problem over data streams can be defined as to monitor k-diverse data objects that exist in the user-specified region and arrived within a time window. Thus, diversification queries are not only parameterized by the diverse subset size k and query range, but also window properties such as window type, size and slide.

The existing work on continuous diversification problem considers only a single query (e.g., [33, 65]). However, in many applications a stream processing system should be able to accommodate a workload of numerous user queries [80, 81], and thus successfully compute the k diverse results at different intervals of time for each of those queries. Therefore, in the future, we plan to focus on the problem of multiple diversification queries over data streams. More specifically, we are interested in monitoring k-diverse data objects for multiple queries

over sliding-window streams. Multi-query processing over streams gives rise to several novel and difficult optimization issues that are very different from those of traditional multi-query optimization. For instance, to extend our existing work on multiple diversification queries to data streams, new techniques for leveraging overlap in data as well as sliding windows need to be investigated. Hence, design of efficient diversification methods based on multiple query optimization techniques for fast retrieval of multiple diverse subsets over dynamic data is a challenging research problem.

References

[1] D. Abadi, P. A. Boncz, S. Harizopoulos, S. Idreos, and S. Madden. The design and implementation of modern column-oriented database systems. *Foundations and Trends in Databases*, 5(3):197–280, 2013.

[2] A. Abouzied, J. M. Hellerstein, and A. Silberschatz. Playful query specification with dataplay. *PVLDB*, 5(12):1938–1941, 2012.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA.*, pages 574–576, 1999.

[4] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. I. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: building fast and reliable approximate query processing systems. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 481–492, 2014.

[5] S. Agarwal, A. Panda, B. Mozafari, S. Madden, and I. Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. *CoRR*, abs/1203.5485, 2012.

[6] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9-11, 2009*, pages 5–14, 2009.

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

[8] A. J. Banday, S. Zaroubi, and M. Bartelmann. *Mining the sky : proceedings of the MPA/ESO/MPE Workshop, held at Garching, Germany, July 31-August 4, 2000*. Heidelberg ; New York : Springer, 2001.

[9] P. Berkhin. A survey of clustering data mining techniques. In *Grouping Multidimensional Data - Recent Advances in Clustering*, pages 25–71. 2006.

[10] T. Bernecker, T. Emrich, F. Graf, H.-P. Kriegel, P. Kröger, M. Renz, E. Schubert, and A. Zimek. Subspace similarity search: Efficient k-nn queries in arbitrary subspaces. In *Scientific and Statistical Database Management, 22nd International Conference, SSDBM 2010, Heidelberg, Germany, June 30 - July 2, 2010. Proceedings*, pages 555–564, 2010.

[11] L. Bin and H. V. Jagadish. Using trees to depict a forest. *PVLDB*, 2(1):133–144, 2009.

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

[13] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430, 2001.

[14] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 335–336, 1998.

[15] U. Çetintemel, M. Cherniack, J. DeBrabant, Y. Diao, K. Dimitriadou, A. Kalinin, O. Papaemmanouil, and S. B. Zdonik. Query steering for interactive data exploration. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[16] K. Chakrabarti, M. Ortega-Binderberger, S. Mehrotra, and K. Porkaew. Evaluating refined queries in top-k retrieval systems. *IEEE Trans. Knowl. Data Eng.*, 16(2):256–270, 2004.

[17] G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, N. Polyzotis, and J. S. V. Varman. The querie system for personalized query recommendations. *IEEE Data Eng. Bull.*, 34(2):55–60, 2011.

[18] H. Chen and D. R. Karger. Less is more: probabilistic models for retrieving fewer relevant documents. In *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, pages 429–436, 2006.

[19] S. Cheng, A. Arvanitis, M. Chrobak, and V. Hristidis. Multi-query diversification in microblogging posts. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014.*, pages 133–144, 2014.

[20] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k-regret minimizing sets. *PVLDB*, 7(5):389–400, 2014.

[21] C. L. A. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*, pages 659–666, 2008.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company, 2003.

[23] A. P. de Vries, N. Mamoulis, N. Nes, and M. L. Kersten. Efficient k-nn search on vertically decomposed data. In *SIGMOD Conference*, pages 322–333, 2002.

[24] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. *DivQ*: diversification for keyword search over structured databases. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, pages 331–338, 2010.

[25] T. Deng and W. Fan. On the complexity of query result diversification. *PVLDB*, 6(8):577–588, 2013.

[26] A. Deshpande and S. Madden. Mauvedb: supporting model-based user views in database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 73–84, 2006.

[27] P. Deshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. Caching multidimensional queries using chunks. In *SIGMOD 1998, Proceedings ACM SIGMOD International*

Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA., pages 259–270, 1998.

[28] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: an automatic query steering framework for interactive data exploration. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 517–528, 2014.

[29] A. Diwan, S. Sudarshan, and D. Thomas. Scheduling and caching in multi-query optimization. In *Proceedings of the International Conference on Management of Data COMAD, Delhi, India 2006.*

[30] M. Drosou and E. Pitoura. Diversity over continuous data. *IEEE Data Eng. Bull.*, 32(4):49–56, 2009.

[31] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 39(1):41–47, 2010.

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

[33] M. Drosou and E. Pitoura. Dynamic diversification of continuous data. In *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, pages 216–227, 2012.

[34] H. Ehsan, M. A. Sharaf, and P. K. Chrysanthis. Muve: Efficient multi-objective view recommendation for visual data exploration. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 731–742, 2016.

[35] E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990.

[36] E. Erkut, Y. Ülküsal, and O. Yeniçerioglu. A comparison of p-dispersion heuristics. *Computers & OR*, 21(10):1103–1113, 1994.

[37] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*, 2001.

[38] J. Fan, G. Li, and L. Zhou. Interactive SQL query suggestion: Making databases user-friendly. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 351–362, 2011.

[39] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 421–432, 2012.

[40] F. Glover, C.-C. Kuo, and K. S. Dhir. Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19(1):109–132, 1998.

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

[42] S. Gollapudi and A. Sharma. An axiomatic framework for result diversification. *IEEE Data Eng. Bull.*, 32(4):7–14, 2009.

[43] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland.*, pages 358–369, 1995.

[44] S. Harizopoulos, V. Liang, D. J. Abadi, and S. Madden. Performance tradeoffs in read-optimized databases. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), Seoul, Korea, September 12-15, 2006*, pages 487–498, 2006.

[45] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 171–182. ACM, 1997.

[46] B. Howe, G. Cole, E. Souroush, P. Koutris, A. Key, N. Khoussainova, and L. Battle. Database-as-a-service for long-tail science. In *Scientific and Statistical Database Management - 23rd International Conference, SSDBM 2011, Portland, OR, USA, July 20-22, 2011. Proceedings*, pages 480–489, 2011.

[47] Z. Huang, Y. Xiang, and Z. Lin. l-skydiv query: Effectively improve the usefulness of skylines. *SCIENCE CHINA Information Sciences*, 53(9):1785–1799, 2010.

[48] Z. Hussain, H. A. Khan, and M. A. Sharaf. Diversifying with few regrets, but too few to mention. In *Proceedings of the Second International Workshop on Exploratory Search in Databases and the Web, ExploreDB 2015, Melbourne, VIC, Australia, May 31 - June 04, 2015*, pages 27–32, 2015.

[49] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012.

[50] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, 2015.

[51] 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.

[52] A. Kalinin, U. Çetintemel, and S. B. Zdonik. Interactive data exploration using semantic windows. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 505–516, 2014.

[53] D. A. Keim and H. Kriegel. Visdb: database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, 14(5):40–49, 1994.

[54] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The researcher's guide to the data deluge: Querying a scientific database in just a few seconds. *PVLDB*, 4(12):1474–1477, 2011.

[55] H. A. Khan, M. Drosou, and M. A. Sharaf. Dos: an efficient scheme for the diversification of multiple search results. In *Conference on Scientific and Statistical Database Management, SSDBM '13, Baltimore, MD, USA, July 29 - 31, 2013*, pages 40:1–40:4, 2013.

[56] H. A. Khan, M. Drosou, and M. A. Sharaf. Scalable diversification of multiple search results. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 775–780, 2013.

[57] H. A. Khan and M. A. Sharaf. Progressive diversification for column-based data exploration platforms. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, 2015.

[58] H. A. Khan, M. A. Sharaf, and A. Albarrak. Divide: efficient diversification for interactive data exploration. In *Conference on Scientific and Statistical Database Management, SSDBM '14, Aalborg, Denmark, June 30 - July 02, 2014*, 2014.

[59] N. Koudas, B. C. Ooi, H. T. Shen, and A. K. H. Tung. LDC: enabling search by partial distance in A hyper-dimensional space. In *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA*, pages 6–17, 2004.

[60] M. J. Kuby. Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987.

[61] C. Li, M. Wang, L. Lim, H. Wang, and K. C. Chang. Supporting ranking and clustering as generalized order-by and group-by. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 127–138, 2007.

[62] L. Li and C. Chan. Efficient indexing for diverse query results. *PVLDB*, 6(9):745–756, 2013.

[63] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *PVLDB*, 2(1):313–324, 2009.

[64] S. Michel, P. Triantafillou, and G. Weikum. KLEE: A framework for distributed top-k query algorithms. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 637–648, 2005.

[65] E. Minack, W. Siberski, and W. Nejdl. Incremental diversification for very large sets: a streaming-based approach. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 585–594, 2011.

[66] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. J. Xu. Regret-minimizing representative databases. *PVLDB*, 3(1):1114–1124, 2010.

[67] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, 1984.

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

[69] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina. Seedb: Visualizing database queries efficiently. *Proc. VLDB Endow.*, 7(4):325–328, 2013.

[70] F. Psallidas, B. Ding, K. Chakrabarti, and S. Chaudhuri. S4: top-k spreadsheet-style search for query discovery. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 2001–2016, 2015.

[71] B. Qarabaqi and M. Riedewald. User-driven refinement of imprecise queries. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 916–927, 2014.

[72] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *PVLDB*, 5(11):1124–1135, 2012.

[73] R. Ramakrishnan and J. Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003.

[74] S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Facility dispersion problems: Heuristics and special cases. In *Workshop on Algorithms and Data Structures*, pages 355–366. Springer, 1991.

[75] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 249–260, 2000.

[76] D. Sacharidis, P. Bouros, and T. K. Sellis. Caching dynamic skyline queries. In *Scientific and Statistical Database Management, 20th International Conference, SSDBM 2008, Hong Kong, China, July 9-11, 2008, Proceedings*, 2008.

[77] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy. Clustering query refinements by user intent. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 841–850. ACM, 2010.

[78] T. Sellam and M. L. Kersten. Meet charles, big data query advisor. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[79] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Balancing energy efficiency and quality of aggregate data in sensor networks. *VLDB J.*, 13(4):384–403, 2004.

[80] M. A. Sharaf, A. Labrinidis, and P. K. Chrysanthis. Scheduling continuous queries in data stream management systems. *PVLDB*, 1(2):1526–1527, 2008.

[81] M. A. Sharaf, A. Labrinidis, P. K. Chrysanthis, and K. Pruhs. Freshness-aware scheduling of continuous queries in the dynamic web. In *Proceedings of the Eight International Workshop on the Web & Databases (WebDB 2005), Baltimore, Maryland, USA, Collocated mith ACM SIGMOD/PODS 2005, June 16-17, 2005*, pages 73–78, 2005.

[82] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 493–504, 2014.

[83] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. Scientific discovery through weighted sampling. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, pages 300–306, 2013.

[84] M. Singh, M. J. Cafarella, and H. V. Jagadish. Dbexplorer: Exploratory search in databases. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 89–100, 2016.

[85] Y. Song, D. Zhou, and L.-w. He. Post-ranking query suggestion by diversifying search results. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 815–824. ACM, 2011.

[86] D. Souravlias, M. Drosou, K. Stefanidis, and E. Pitoura. On novelty in publish/subscribe delivery. In *Workshops Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 20–22, 2010.

[87] K. Stefanidis, M. Drosou, and E. Pitoura. Perk: personalized keyword search in relational databases through preferences. In *EDBT 2010, 13th International Conference on*

Extending Database Technology, Lausanne, Switzerland, March 22-26, 2010, Proceedings, pages 585–596, 2010.

[88] C. Stolte and P. Hanrahan. Polaris: A system for query, analysis and visualization of multi-dimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8:52–65, 2002.

[89] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-store: A column-oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 553–564, 2005.

[90] D. F. Swayne, D. T. Lang, A. Buja, and D. Cook. Ggobi: evolving from xgobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 43(4):423–444, 2003.

[91] A. Tamir. Obnoxious facility location on graphs. *SIAM Journal on Discrete Mathematics*, 4(4):550–567, 1991.

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

[93] F. Tauheed, T. Heinis, F. Schürmann, H. Markram, and A. Ailamaki. SCOUT: prefetching for latent feature following queries. *PVLDB*, 5(11):1531–1542, 2012.

[94] Q. T. Tran, C. Chan, and S. Parthasarathy. Query by output. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 535–548, 2009.

[95] G. Valkanas, A. N. Papadopoulos, and D. Gunopulos. Skydiver: a framework for skyline diversification. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 406–417, 2013.

[96] E. Vee, J. Shanmugasundaram, and S. Amer-Yahia. Efficient computation of diverse query results. *IEEE Data Eng. Bull.*, 32(4):57–64, 2009.

[97] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. On query result diversification. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 1163–1174, 2011.

[98] C. Weaver. Building highly-coordinated visualizations in improvise. In *10th IEEE Symposium on Information Visualization (InfoVis 2004), 10-12 October 2004, Austin, TX, USA*, pages 159–166, 2004.

[99] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, pages 368–378, 2009.

[100] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. Recommendation diversification using explanations. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 1299–1302, 2009.

[101] C. Zhai, W. W. Cohen, and J. D. Lafferty. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *SIGIR 2003: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 28 - August 1, 2003, Toronto, Canada*, pages 10–17, 2003.

[102] J. Zhang and C. Chen. Collaboration in an open data escience: A case study of sloan digital sky survey. *CoRR*, abs/1001.3663, 2010.

[103] C. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 22–32, 2005.

[104] K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1555–1566, 2014.

Appendix A

An Appendix

A.1 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.